# Appunti di Calcolo Numerico

con codici in Matlab/Octave

Stefano De Marchi
Dipartimento di Informatica,
Università di Verona

November 25, 2009

## Introduzione

Queste pagine sono gli appunti del corso di Calcolo Numerico che il sottoscritto ha tenuto dall'AA. 2006-07, per il corso di laurea triennale in Matematica Applicata e Informatica Multimediale della Facoltà di Scienze dell'Università degli Studi di Verona. Al lettore è richiesta la familiarità con `Matlab`, `MATrix LABoratory`, o la sua versione freeware `Octave`, di cui si è spesso fatto uso nel testo per scrivere pezzi di codici che implementano alcuni degli esempi e algoritmi numerici. Chi desiderasse conoscere Matlab, la sua sintassi e il suo utilizzo, rimandiamo alla lettura del libro [19] oppure al manuale disponibile in rete all'indirizzo

`www.ciaburro.it/matlab/matlab.pdf`.

Per quanto riguarda `Octave`, il manuale è incluso nel download del package che si trova al link

`http://www.gnu.org/software/octave/`.

La versione disponibile più recente di Octave è la 2.9.14 che è una pre-release di Octave 3.0 (dati: ottobre 2007).

Gli appunti sono organizzati in 6 capitoli, corrispondenti anche agli argomenti fondamentali trattati in un corso di base di Calcolo Numerico:

- Cap. 1: Rappresentazione dei numeri e analisi degli errori.
- Cap. 2: Ricerca di zeri di funzione.
- Cap. 3: Soluzione di sistemi lineari.
- Cap. 4: Calcolo di autovalori di matrici.
- Cap. 5: Interpolazione e approssimazione.
- Cap. 6: Derivazione e integrazione.

In ogni capitolo c'è una sessione di *Esercizi proposti*: si tratta di una raccolta di esercizi proposti nel corso degli ultimi tre anni nei vari appelli, compiti e compitini da parte dell'autore. Pertanto per la maggior parte di essi si possono trovare le soluzioni e, dove richiesto, il codice Matlab, andando alla pagina web

`http://profs.sci.univr.it/~demarchi/didattica.html`.

Il testo non ha la pretesa di essere sostitutivo di libri molto più completi e dettagliati disponibili in letteratura, come ad esempio i libri [1, 4, 5, 15, 18, 19, 20, 22], ma come traccia

di riferimento per un corso di Calcolo Numerico. Pertanto l'invito è di consultare anche i testi citati in bibliografia, sia per cultura personale, ma soprattutto per un completamento della preparazione.

Ringrazio fin d'ora tutti coloro che mi segnaleranno sviste ed errori e mi daranno dei consigli per eventuali miglioramenti.

Stefano De Marchi
Dipartimento di Informatica
Università di Verona.

# Indice

# Elenco delle Figure

# Elenco delle Tabelle

# Capitolo 1

# Rappresentazione dei numeri e analisi degli errori

In questo capitolo iniziale, metteremo per così dire le basi per comprendere la *filosofia* sottostante al calcolo numerico. L'analisi degli errori è fondamentale per comprendere come evitarli, ma se non fosse possibile evitarli, come ridurli almeno al minimo possibile.

Ma per comprendere quali sono i tipi d'errore di cui dobbiamo tenere conto, prima di tutto dobbiamo capire come si rappresentano i numeri in un calcolatore. Vedremo che la rappresentazione dei numeri è una delle fonti principali d'errore detti appunti *errori di rappresentazione*.

## 1.1 Rappresentazione dei numeri in un calcolatore

La notazione che maggiormente si usa nei calcolatori è la notazione a *virgola mobile* o in inglese *floating-point*. Se $a$ è un numero, intero o reale, usando la notazione a virgola mobile, lo possiamo scrivere come

$$a = pN^q \,, \tag{1.1}$$

dove $p$ si chiama `mantissa` che è un numero reale, $N$ è la `base di numerazione` (solitamente $N = 2$, base binaria) e $q$ è un intero che si chiama `esponente`.

Osserviamo anzitutto che la *notazione non è unica*. Infatti

$$a = pN^q = p_1 N^{q-1} = p_2 N^{q+1}$$

con $p_1 = Np$ e $p_2 = p/N$.

Se la mantissa $p$ è tale che

$$\frac{1}{N} < |p| < 1$$

allora la rappresentazione (1.1) si dice *normalizzata*. Facciamo due esempi

- $a = 115.78$, la sua forma normalizzata è $a = 0.11578 \cdot 10^3$.
- $a = 0.0026$, la sua forma normalizzata è $a = 0.26 \cdot 10^{-2}$.

Pertanto, fissata la base di numerazione $N$, per la rappresentazione di un numero $a$ dovremo conoscere la coppia $(p, q)$ (mantissa ed esponente). Nel caso $a = 0$, $(p, q) = (0, 0)$. In generale si usa la seguente rappresentazione dove `s` indica il bit riservato al segno del numero

| s | $q$ | $\lvert p\rvert$ |
|---|---|---|

Tabella 1.1: Rappresentazione dei numeri in un calcolatore

e che assume valori $s = 0$ se il segno è $+$ e $s = 1$ quando il segno è $-$; $q$ lo spazio per l'esponente e $|p|$ lo spazio per la mantissa normalizzata.

**Definizione 1.** *Si chiama* `numero macchina` *un numero tale che $p$ e $q$ sono rappresentabili esattamente negli spazi riservati.*

Se ad esempio, lo spazio per $|p|$ è formato da $t$ cifre, i numeri macchina sono tutti quelli che hanno la mantissa normalizzata con non più di $t$ cifre. Per l'esponente valgono le disuguaglianze

$$m \leq q \leq M$$

dove il minimo $m < 0$ e il massimo $M > 0$ dipendono da calcolatore a calcolatore. Posto $q^* = q - m \geq 0$ allora

$$0 \leq q^* \leq M - m \, .$$

Parleremo poi di `singola precisione` se la rappresentazione di Tabella 1.1 è su 32bits (essendo 1byte=8bits essa equivale a 4 bytes) (cfr. Tabella 1.2), di `doppia precisione` quando la rappresentazione di Tabella 1.1 è su 64bits (8 bytes) (cfr. Tabella 1.3). Nel

| 1 S 1 | 2 $q$ 9 | 10 $\lvert p\rvert$ 32 |
|---|---|---|

Tabella 1.2: Rappresentazione in singola precisone: i numeretti indicano i bits d'inizio e fine delle parti corrispondenti al segno, esponente e mantissa.

caso di singola precisione, essendoci 8 bits riservati all'esponente, allora $2^8 - 1 = 255$ sarà il massimo numero rappresentabile. Da cui, essendo $0 \leq q^* \leq 255$ dalla relazione $q^* = q - m$ avremo che $-127 \leq q \leq 128$.

| 1 | S | 1 | 2 | $q$ | 12 | 13 | $\lvert p\rvert$ | 64 |
|---|---|---|---|---|---|---|---|---|

Tabella 1.3: Rappresentazione in doppia precisone: i numeretti, come in Tabella 1.2 indicano i bits d'inizio e fine delle parti.

Lo standard ANSI/IEEE 754-1985 (modificato nel 1989 e ufficialmente detto *IEC 60559:1989, binary floating-point arithmetic for microprocessor systems*) usa, per la mantissa normalizzata, la rappresentazione

$$p = \pm 1.a_{-1}a_{-2}\ldots a_{-23}\ .$$

Avremo

- L'esponente $q^* = 0$, corrispondente a $q = -127$, viene usato per la codifica dello zero $p = 0$ ed eventuali numeri non normalizzati, quali $p = \pm 0.a_{-1}a_{-2}\ldots a_{-23}$ .
- L'esponente $q^* = 255$ è riservato per numeri non rappresentabili. Di solito questo viene identificato con `NaN` (Not a Number), ad esempio se si ha $\infty$ oppure un valore non definito quale $\log(-2)$.
- I restanti valori dell'esponente sono usati per codificare la mantissa normalizzata. In singola precisione, si tratta dei numeri che hanno esponente $-126 \le q \le 127$ mentre in doppia $-1023 \le q \le 1024$.

Pertanto, usando questo standard, il generico numero di macchina in singola precisione ha la forma

$$\begin{aligned}(-1)^s p\, 2^{q^*-127},\ \ 1 \le q^* \le 254\\ p = \pm 1.a_{-1}a_{-2}\ldots a_{-23}\ ,\end{aligned}$$

mentre in doppia precisione, avendo 3 bit in più per l'esponente e 29 in più per la mantissa

$$\begin{aligned}(-1)^s p\, 2^{q^*-1023},\ \ 1 \le q^* \le 2046\\ p = \pm 1.a_{-1}a_{-2}\ldots a_{-52}\ ,\end{aligned}$$

Esempio 1. Supponiamo di volere rappresentare in singola precisione il numero decimale $a = 43.6875$ nello standard ANSI/IEEE 784-1985.

(i) Dapprima dovremo trasformare il numero, senza segno, in forma binaria. La parte intera, 43, diventa, $43_{10} = 101011_2$. La parte frazionaria $0.6875_{10} = 1011_2$. Complessivamente $43.6875_{10} = 101011.1011_2$.

(ii) Successivamente spostiamo la virgola verso sinistra, lasciando solo un 1 alla sinistra: $1.010111011 \cdot 2^5$ .

La mantissa è quindi riempita con zeri a destra, fino a completare i 23 bit, ottenendo 1.01011101100000000000000.

(iii) L'esponente è 5, ma dobbiamo convertirlo in forma binaria e adattarlo allo standard. Per la singola precisione, dobbiamo aggiungere 127 (detto anche `bias`), ovvero $5 + 127 = 132_{10} = 10000100_2$.

Al link `http://babbage.cs.qc.edu/IEEE-754/Decimal.html`, il lettore interessato, troverà un'interessante interfaccia che consente di trasformare numeri decimali in numeri binari (ed esadecimali) proprio nello standard IEEE 784.

Come ultima nota, lo standard è attualmente sotto revisione (IEEE 754r) e i lavori di revisione, il cui termine era previsto per l'anno 2005, non sono ancora stati conclusi. Per maggiori informazioni, si rimanda al link `http://it.wikipedia.org/wiki/IEEE_754`.

## 1.2 Analisi degli errori

Sia $x$ un numero che rappresenta un valore esatto. Indichiamo con $\tilde{x}$ una sua rappresentazione sul calcolatore. Allora

$$\begin{aligned} E_a &:= |x - \tilde{x}| , \\ E_{r_x} &:= \left| \frac{x - \tilde{x}}{x} \right| , \quad x \neq 0 \\ E_{r_{\tilde{x}}} &:= \left| \frac{\tilde{x} - x}{\tilde{x}} \right| , \quad \tilde{x} \neq 0 , \end{aligned}$$

definiscono l'`errore assoluto`, `errore relativo su` $x$ e l'`errore relativo su` $\tilde{x}$, rispettivamente.

Usando l'espressione di $E_{r_x}$ possiamo scrivere anche

$$\tilde{x} = x(1 + \epsilon) , \quad \epsilon = \frac{\tilde{x} - x}{x} , \tag{1.2}$$

che ci dice come un'approssimazione si ottenga dal valore esatto a meno di un errore di rappresentazione. Questo errore, si dice `errore inerente` o `ineliminabile` poiché esso dipende dalla rappresentazione (finita) dei numeri su un calcolatore.

Nel caso di un sistema floating-point in base $N$ con mantissa a cui sono riservate $t$ posizioni o cifre, tutti i numeri che nella rappresentazione normalizzata hanno più di $t$ cifre

(con esponente $m \leq q \leq M$) dovranno venire approssimati. Come? Ci sono sostanzialmente due tipi di approssimazione a cui corrispondono anche analoghi errori di rappresentazione.

(a) `troncamento`: della mantissa $p$ del numero, si prendono solo $t$ cifre, le altre dalla $t+1$-esima in poi non si considerano. Ad esempio se $p = 0.7243591$, $N = 10$ e $t = 5$, allora $\tilde{p} = 0.72435$.

(b) `arrotondamento`: alla cifra $t$-esima della mantissa $p$ viene aggiunta la quantità 0.5 e poi si opera come in (a). Nell' esempio di prima, alla quinta cifra di $p = 0.7243591$, che è 5, si somma 0.5 che diventa 6, cosicché $\tilde{p} = 0.72436$.

Tra le due tecniche, troncamento e arrotondamento, qual è quella che consente di commettere un errore inferiore?

Dato un numero $a = pN^q$ indichiamo con $\tilde{a} = \tilde{p}N^q$ una sua approssimazione. Osserviamo che le mantisse $\tilde{p}$ dei numeri macchina $1/N \leq \tilde{p} < 1$ non hanno più di $t$ cifre e la distanza tra due mantisse consecutive $p_1, p_2$ è proprio $N^{-t}$, da cui $|p - p_1| < N^{-t}$ (analogamente per $p_2$). Vediamo cosa accade degli errori nei casi di troncamento(a) e di arrotondamento (b).

(a)
$$|a - \tilde{a}| = |(p - \tilde{p})|N^q < N^{q-t}$$
essendo $p$ e $\tilde{p}$ consecutive.

(b)
$$|a - \tilde{a}| = |(p - \tilde{p})|N^q \leq \frac{1}{2}N^{q-t}$$
essendo $p$ e $\tilde{p}$ consecutive ma nel caso di arrotondamento $|p - \tilde{p}| \leq \frac{1}{2}N^{-t}$.

Segue che l'*approssimazione per arrotondamento è da preferirsi*! Infine, per quanto rigurdano i corrispondenti errori relativi si ha:

(a)
$$\frac{|a - \tilde{a}|}{|a|} < N^{1-t},$$
poiché, essendo $N^{q-1} < |a| < N^q$ e dal fatto che $|a - \tilde{a}|/|a| < N^{q-t}/N^{q-1}$, si ottiene la maggiorazione di cui sopra.

(b)
$$\frac{|a - \tilde{a}|}{|a|} \leq \frac{1}{2}N^{1-t}.$$

A questo punto vale la seguente definizione

**Definizione 2.** *Il numero*

$$\texttt{eps} = \frac{1}{2}N^{1-t}\,, \tag{1.3}$$

*si chiama* `precisione macchina`.

In pratica, la precisione macchina, rappresenta quella costante caratteristica di ogni aritmetica (arrotondata) floating-point ed è la massima precisione con cui vengono effettuati i calcoli su quella particolare macchina. Detto altrimenti, `eps` è il più piccolo numero che sommato a 1 dà un numero maggiore di 1. Pertanto un algoritmo, scritto in codice Matlab/Octave, per il calcolo di `eps` con $N = 2$ in doppia precisione è il seguente:

```
e=1; k=0;
while (e+1 > 1)
  e=e/2; k=k+1;
end
e=2*e %e' necessario perche' si era diviso per 2
k-1     % mi da l'esponente
```

dove il contatore `k` serve a ricordare il numero di divisioni e indica pure l'esponente della rappresentazione del numero `eps`. La moltiplicazione finale è necessaria perché dopo che il test è stato verificato, `e` avrebbe un valore metà del valore vero. Se ora facciamo eseguire il codice, otterremo il seguente risultato

```
e = 2.2204e-016
k = 52
```

infatti $\texttt{e} = 2^{-52}$. Vale la pena ricordare che in Matlab/Octave esiste la costante predefinita `eps` il cui valore è appunto `2.2204e-016`.

## 1.3 Operazioni con numeri macchina

Se indichiamo con con $fl(a) = \tilde{a}$ l'operazione di arrotondamento e con $\oplus$, $\ominus$, $\odot$ e $\oslash$ le corrispondenti operazioni aritmetiche fatta sui numeri macchina, valgono per esse le seguenti regole

$$\begin{aligned}
\tilde{a} \oplus \tilde{b} &= fl(\tilde{a} + \tilde{b}) = (\tilde{a} + \tilde{b})(1 + \epsilon_1) \\
\tilde{a} \ominus \tilde{b} &= fl(\tilde{a} - \tilde{b}) = (\tilde{a} - \tilde{b})(1 + \epsilon_2) \\
\tilde{a} \odot \tilde{b} &= fl(\tilde{a} \cdot \tilde{b}) = (\tilde{a} \cdot \tilde{b})(1 + \epsilon_3) \\
\tilde{a} \oslash \tilde{b} &= fl(\tilde{a}/\tilde{b}) = (\tilde{a}/\tilde{b})(1 + \epsilon_4)
\end{aligned}$$

con $|\epsilon_i| < \texttt{eps}$.

La domanda da porsi è se per queste operazioni macchina valgono le stesse regole che per le corrispondenti operazioni aritmetiche. La risposta è in generale *negativa*.

ESEMPIO 2. Consideriamo la somma di due numeri floating-point. Infatti $\tilde{a} \oplus \tilde{b} = \tilde{a}$ se $0 < |\tilde{b}| \ll |\tilde{a}|$

Facciamo vedere un esempio che anche per numeri macchina si possono presentare dei problemi.

ESEMPIO 3. Siano $a = p_1 N^{q_1}$ e $b = p_2 N^{q_2}$. Consideriamo $a \oslash b$. Il risultato sarà `overflow` (esponente maggiore di $M$) se $q_1 > 0$, $q_2 < 0$ e $q_1 - q_2 > M$ oppure `underflow` (esponente minore di $m$) se $q_1 < 0$, $q_2 > 0$ e $q_1 - q_2 < m$.

A conferma ulteriore dei problemi che si possono verificare lavorando con numeri macchina, diamo alcuni semplici esercizi.

ESERCIZIO 1. *Calcolare l'espressioni* $a+(b+c)$ *e* $(a+b)+c$ *dove* $a = 1.0e{+}308$, $b = 1.1e{+}308$ *e* $c = -1.001e + 308$.

ESERCIZIO 2. *Sia* $x = 1.0e-15$. *Calcolare* $\dfrac{(1+x)-1}{x}$ . *Perchè l'espressione è inaccurata?*

ESERCIZIO 3. *Si consideri il polinomio*

$$f(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1\,.$$

*Lo si valuti su 401 punti equispaziati per* $x \in [1 - 2 \cdot 10^{-8}, 1 + 2 \cdot 10^{-8}]$. *Si plotti quindi il grafico* $(x, f(x))$ *e il grafico di* $(x, p(x))$ *con* $p(x) = (x-1)^7$, *sugli stessi punti. Se ne discutano i risultati.*

Uno dei problemi che maggiormente si presentano negli algoritmi numerici è la *cancellazione numerica* che in sostanza è la *perdita di cifre significative*.

Anzitutto comprendiamo che cosa sono le cifre significative di un numero. Ad esempio 13020.0 ha cifre significative 1302 mentre 0.0534 ha cifre significative 534.

Se due numeri sono quasi uguali, dove *uguali* s'intende a meno della precisione macchina, allora è possibile il verificarsi della cancellazione numerica. Vediamo alcuni esempi.

ESEMPIO 4. Consideriamo i numeri $a = p_1 N^q$ con $p_1 = 0.147554326$ e $b = p_2 N^q$ con $p_2 = 0.147251742$ e $N = 10$. In aritmetica a $t = 6$ cifre significative, avremo $\tilde{p_1} = 0.147554$ e $\tilde{p_2} = 0.147252$. Ora $a - b = (p_1 - p_2)N^q = (p_1 - p_2)10^3 = 0.302584$. Ma $(\tilde{p_1} \ominus \tilde{p_2})10^3 = 0.302000$ con la perdita delle cifre significative 584.

ESEMPIO 5. Consideriamo il calcolo della funzione $f(x) = \dfrac{e^x - 1}{x}$ in un punto $x_0$. La funzione data si può anche vedere come la serie $\displaystyle\sum_{k=2}^{\infty} \frac{x^{i-1}}{i!}$. Pertanto si possono usare due algoritmi per il calcolo di $f(x_0)$

```
ALGORITMO 1

if x0==0
 f=1;
else
 f=(exp(x0)-1)/x0;
end
```

```
ALGORITMO 2

y=exp(x0);
if y==1,
   f=1;
else
   f=(y-1)/log(y);
end
```

Nel caso in cui $|x| \ll 1$ (cioè molto vicino a 0, usando i due algoritmi otterremo i seguenti risulati

| $x_0$ | ALG.1 | ALG. 2 |
|---|---|---|
| $1.e-5$ | 1.000005 | 1.000005 |
| $1.e-6$ | 1.0036499 | 1.0000005 |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $1.e-15$ | 1.1102... | 1.000....000 (15 zeri) |
| $1.e-16$ | 0 | 1 |

Pertanto l'ALGORITMO 2 è più *stabile* (chiariremo meglio più avanti il concetto di stabilità di un algoritmo numerico). Infatti, nell'ipotesi di singola precisione, la risposta esatta sarebbe 1.00000005. Se infatti consideriamo $fl((e^x - 1)/x) \approx 1.3245....$ mentre $fl((e^x - 1)/(\log(e^x)) \approx 1.00000006$ che è la risposta corretta.

**Cosa fare per evitare la cancellazione numerica?** Una prima risposta è di trovare un'espressione più stabile, ovvero tale da non far aumentare gli errori introdotti dalla formulazione del problema.

Ad esempio, si voglia valutare $\sqrt{x+\delta} - \sqrt{x}$ per $\delta \to 0$. Razionalizzando si ottiene $\dfrac{\delta}{\sqrt{x+\delta} + \sqrt{x}}$ dove si evitano i problemi di cancellazione che si avrebbero con l'espressione originale.

Un altro esempio è il calcolo di $\cos(x+\delta) - \cos(x)$ sempre per $\delta \to 0$. Qui possiamo evitare i problemi di cancellazione usando la formula di *prostaferesi della differenza di coseni*: $\cos(x+\delta) - \cos(x) = -2\sin(\delta/2)\sin(x+\delta/2)$.

Come ultimo esempio, consideriamo di valutare $f(x) = x(x - \sqrt{x^2-1})$ quando $x \to +\infty$. Infatti per un tale valore $\sqrt{x^2-1} \approx x$. Pertanto, sempre razionalizzando possiamo scrivere $f(x) = \dfrac{x}{\sqrt{x^2-1}+x}$ evitando i soliti problemi di instabilità dovuti alla cancellazione.

## 1.4 Stabilità e condizionamento

Iniziamo subito con la definizione di stabilità di un metodo numerico.

**Definizione 3.** *Un metodo numerico (formula, algoritmo) si dice* `stabile` *se non propaga gli errori. Altrimenti si dice* `instabile`*.*

La stabilità è quindi un concetto legato al metodo risolutivo ovvero al corrispondente algoritmo. Lo scopo dell'analisi di stabilità è di capire come avviene la propagazione degli errori. Se questa è controllata, cioè non li fa crescere, allora il metodo sarà stabile. Uno dei problemi connessi all'instabilità è la cancellazione numerica, proprio come evidenziato nei due esempi successivi.

Esempio 6. Desideriamo risolvere l'equazione $ax^2 + bx + c = 0$. Se $a \neq 0$, le radici sono $x_{1,2} = \dfrac{-b \pm \sqrt{b^2-4ac}}{2a}$. Dove si manifesta la cancellazione numerica?

- In $x_1$ quando $\sqrt{b^2-4ac} \approx b$
- in $x_2$ quando $-\sqrt{b^2-4ac} \approx b$.

Come ovviare a questi problemi? Nel primo caso, prima si calcola $x_2$ dove il problema della cancellazione non sussiste quindi, usando le relazioni tra le radici $x_1$ e $x_2$ di un'equazione di secondo grado, otteniamo $x_1 = c/(ax_2)$. In maniera analoga opereremo nel secondo caso: prima calcolo $x_1$ quindi $x_2 = c/(ax_1)$.

Esempio 7. Data $f(x) = x^2$ si voglia calcolare $f'(x)$ per $x = x_0$. Ora, ricorrendo alla definizione di derivata come

$$\lim_{h\to 0} \frac{f(x+h)-f(x)}{h} \quad \text{per } x = x_0 \,,$$

ma per $h \to 0$ potrebbero insorgere problemi di cancellazione. Cosa che si ovvia ricorrendo alla relazione $f'(x) = 2x$ che verrà quindi valutata per $x = x_0$.

Riassumendo, la stabilità è legata al metodo risolutivo e l'instabilità è dovuta essenzialemente agli *errori algoritmici* legati alle operazioni da effettuarsi durante l'esecuzione

dell'algoritmo. Ma non dimentichiamo gli errori di rappresentazione (che sono errori inevitabili).

L'altro aspetto da tenere presente nell'analisi è quello che definiremo come *condizionamento* del problema numerico. Questo aspetto è legato alla definizione del problema, matematicamente una funzione dei dati del problema. Una definizione che spesso troviamo nei testi è la seguente.

**Definizione 4.** *Un problema si dice* `ben condizionato` *se a piccole perturbazioni (relative) sui dati in ingresso corrispondono perturbazioni (relative) dello stesso ordine in uscita. In caso contrario il problema si dice* `mal condizionato`.

Per misurare il condizionamento si introduce il cosidetto *numero di condizionamento*

$$C = \frac{r}{d}\,, \tag{1.4}$$

dove $r$ indica la percentuale d'errore sul *risultato* rispetto alla percentuale d'errore sul *dato* $d$. Pertanto, usando questo indice, un problema sarà ben condizionato quando $C$ è piccolo (vedremo più oltre in che senso) altrimenti sarà mal condizionato. Vediamo un esempio.

ESEMPIO 8. Il sistema

$$\begin{cases} x + y = 2 \\ 1001x + 1000y = 2001 \end{cases}$$

ha soluzione $(x, y) = (1, 1)$. Siano

$$A = \begin{pmatrix} 1 & 1 \\ 1001 & 1000 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 2 \\ 2001 \end{pmatrix}.$$

Ora, perturbiamo l'elemento $a_{1,1}$ della matrice $A$ di 0.01, ovvero consideriamo la matrice

$$A_1 = A + \begin{pmatrix} 0.01 & 0 \\ 0 & 0 \end{pmatrix}.$$

Se risolviamo il sistema $A_1\mathbf{x} = \mathbf{b}$ otteniamo la soluzione $(x, y) = (-1/9, 1901/900)$. Pertanto, per calcolare il numero di condizionamento (1.4), dobbiamo vedere chi sono i rapporti, $r/d$, su ogni componente del vettore soluzione:

$$\left(\frac{r}{d}\right)_x = \frac{1 - (-1/9)}{1} = 1.\bar{1}, \quad \left(\frac{r}{d}\right)_y = \frac{1 - (1901/900)}{1} = -1.11\bar{2}$$

da cui, complessivamente in percentuale, $C = 111\%$. Quindi, un errore di $10^{-2}$ sul dato $A$ (misurato in qualunque norma matriciale) si è riversato con un errore di circa $10^2$ sul risultato. Il problema è quindi *mal condizionato*.

Consideriamo la valutazione di una funzione $f : \mathbb{R} \to \mathbb{R}$ in un punto $x_0$. Prendiamo ora una perturbazione $x_0 + h$. Le quantità $r$ e $d$ richieste in (1.4), in questo caso sono

$$d = \frac{x_0 + h - x_0}{x_0} = \frac{h}{x_0}\,; \quad r = \frac{f(x_0 + h) - f(x_0)}{f(x_0)}\,,$$

da cui

$$C(f, h) := \frac{f(x_0 + h) - f(x_0)}{h}\,\frac{x_0}{f(x_0)}\,.$$

Al tendere di $h \to 0$,

$$\lim_{h\to 0} |C(f, h)| = |C(f, x_0)| = \left| f'(x_0) \cdot \frac{x_0}{f(x_0)} \right| \,.$$

Questo esempio ci dice che il numero di condizionamento tende ad un limite che in modulo vale

$$|C(f, x_0)| = \left| f'(x_0) \cdot \frac{x_0}{f(x_0)} \right| \,,$$

che viene detto `fattore d'amplificazione d'errore`. Se $C(f, x_0) < 1$ diremo che il problema è ben condizionato altrimenti verrà detto malcondizionato.

Come applicazione di quest'analisi, consideriamo $f(x) = \sqrt{1 - x}$. Ora $f'(x) = -\frac{1}{2\sqrt{1-x}}$ e quindi

$$C(f(x)) = \left| \frac{x}{2(1 - x)} \right|$$

che ha problemi quando $x \approx 1$. Ad esempio se $x = 0.999$ e $h = 10^{-5}$ allora

$$d = h/x = 1.001 \cdot 10^{-5}, \quad r = \frac{\sqrt{1 - (x + h)} - \sqrt{1 - x}}{\sqrt{1 - x}} \approx -0.00503$$

da cui $\left|\frac{r}{d}\right| \approx 501.67$. Anche passando al limite per $h \to 0$ le cose non migliorano. Il problema è malcondizionato e ciò è dovuto al fatto che il fattore d'amplificazione richiede il calcolo della derivata. Questo ci dice anche che il calcolo della derivata è un problema, in genere, malcondizionato.

## 1.5 Il calcolo di $\pi$

Per il calcolo di $\pi$ esistono alcuni importanti algoritmi non tutti convergenti per motivi di instabilità. Di seguito diamo cenno di 5 algoritmi tra i più importanti. Di essi diamo anche un *pseudo-algoritmo* che è un'utile base di partenza per una successiva implementazione in un linguaggio di programmazione.

1. *Algoritmo di Archimede.* Mediante questo algoritmo, $\pi$ è approssimato con l'*area* del poligono regolare di $2^n$ lati inscritto nella circonferenza di raggio 1 (che ha area uguale a $\pi$).

   Indicando con $b_i$ il numero di lati dell'$i$-esimo poligono regolare iscritto, con $s_i = \sin\left(\frac{\pi}{2^i}\right)$ e con $A_i$ la corrispondente area, l'algoritmo si può così descrivere:

   **Algoritmo**

   $b_1 = 2;\ s_1 = 1$
   `for i=2:n`
   $A_i = b_{i-1}s_{i-1},\ s_i = \sqrt{\frac{1-\sqrt{1-s_{i-1}^2}}{2}}$
   $b_i = 2b_{i-1}$
   `end for`

2. *Algoritmo di Viète.* Mediante questo algoritmo, $\pi$ è approssimato con il *semi-perimetro* del poligono regolare di $2^n$ lati inscritto nella circonferenza di raggio 1.

   Indicando con

   $$c_i = \cos\left(\frac{\pi}{2^i}\right)$$

   e $p_i$ il corrispondente semiperimetro, l'algoritmo si descrivere come segue:

   **Algoritmo**

   $c_1 = 0;\ p_1 = 2$
   `for i=2:n`
   $c_i = \sqrt{\frac{1+c_{i-1}}{2}}$
   $p_i = \frac{p_{i-1}}{c_i}$
   `end for`

3. *Algoritmo di Wallis.* Qui $\pi$ è approssimato con la formula:

   $$\frac{\pi}{2} = \frac{2}{1}\frac{2}{3}\ \frac{4}{3}\frac{4}{5}\ \cdots\ \frac{2n}{2n-1}\frac{2n}{2n+1}\ \cdots \quad n \geq 1\ .$$

   Indicando con $p_i$ la produttoria al passo $i$, l'algoritmo si descrivere come segue:

   **Algoritmo**

   $p_0 = 2;$
   `for i=1:n,`
   $p_i = p_{i-1}\frac{4i^2}{4i^2-1};$
   `end for`

4. $\pi = 4\ \arctan(1)$. Usando l'espansione di Taylor di $\arctan(1)$, $\pi$ è approssimato con la formula:

   $$\arctan(1) = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7}\ \cdots\right).$$

   Indicando con $q_i$ la somma al passo $i$, l'algoritmo si può descrivere come segue:

**Algoritmo**

$q_1 = 1$;
`for i=2:n`
$q_i = q_{i-1} + \frac{(-1)^{i-1}}{2i-1}$
`end for`

5. $\pi = 6\ \arcsin(\frac{1}{2})$. Come nel caso dell' arctan, $\pi$ è approssimato con la seguente formula che si ottiene ancora una volta espandendo in serie di Taylor l' $\arcsin(\frac{1}{2})$:

$$\arcsin\left(\frac{1}{2}\right) = 6\left(\frac{1}{2} + \frac{1}{2}\frac{1}{3}\frac{1}{2^3} + \frac{1}{2}\frac{3}{4}\frac{1}{5}\frac{1}{2^5} + \cdots\right).$$

Indicando con $q_i$ la somma al passo $i$ e $t_i$ il "punto" corrente, l'algoritmo si descrivere come segue:

**Algoritmo**

$q_1 = 0;\ t_1 = \frac{1}{2}$
`for i=1:n-1`
$q_{i+1} = q_i + \frac{t_i}{2i-1};\ \ t_{i+1} = \frac{t_i(2i-1)}{8i}$
`end for`
$\pi = 6q_n$

## 1.6 Esercizi proposti

ESERCIZIO 4. (`Laboratorio del 19/10/05`). *Implementare queste operazioni:*

1. *a=4/3;*
2. *b=a-1;*
3. *c=b+b+b;*
4. *e=1-c.*

*Qual è il risultato dell'operazione?*

ESERCIZIO 5. (`Laboratorio del 19/10/05`). *Siano $x = 5$ e $y = 5 - \eta$ con $x - y = \eta$. L'errore relativo della differenza è*

$$\epsilon_{x-y} = \frac{fl(x-y) - (x-y)}{x-y},$$

*dove $fl(x-y)$ è la differenza dei 2 numeri $x$ e $y$, in aritmetica floating point. Ovvero*

$$fl(x-y) = (x-y)(1+\texttt{eps}),$$

*con* `eps` *la funzione Matlab/Octave che restituisce la precisione macchina. Calcolare $\epsilon_{x-y}$ al diminuire di $\eta$ e riportare su una tabella i valori $\eta$, $\epsilon_{x-y}$ e la percentuale $\epsilon_{x-y} * 100$.*

ESERCIZIO 6. `(Laboratorio del 19/10/05)`. *Si consideri la ricorrenza*

$$\begin{aligned} z_2 &= 2\,, \\ z_{n+1} &= \sqrt{2}\,\frac{z_n}{\sqrt{1+\sqrt{1-4^{1-n}\,z_n^2}}}\,; n \geq 2 \end{aligned}$$

*che converge a $\pi$ quando $n \to \infty$. Scrivere un M-file che implementa la ricorrenza precedente e inoltre visualizza in scala logaritmica al variare di $n$ l'errore relativo $\frac{|\pi - z_n|}{\pi}$.*

*La formula ricorrente è stabile?*

ESERCIZIO 7. *Si consideri la ricorrenza*

$$\begin{aligned} I_0 &= \frac{1}{e}(e-1) = \frac{1}{e}\int_0^1 x^0 e^x dx\,, \\ I_{n+1} &= 1-(n+1)I_n = \frac{1}{e}\int_0^1 x^{n+1} e^x dx\,; n \geq 0 \end{aligned}$$

*sapendo che $I_n \to 0$ per $n \to \infty$, si scriva un M-file che calcola $I_{40}$. La ricorrenza è stabile? Come è possibile stabilizzarla?* `Sugg.` *Si può procedere mediante stabilizzazione all'indietro. Ovvero posto n=40, si calcola*

$$\begin{aligned} v_n &= \frac{1}{e}\int_0^1 x^n e^x dx\,, \\ v_{i-1} &= (1-v_i)/i, \quad i = n, n-1, \ldots, 2 \end{aligned}$$

*Per il calcolo di $v_n$ usare la funzione Matlab/Octave* `quadl` *con la seguente chiamata* `quadl('f',0,1,[],n)`.

ESERCIZIO 8. *Si consideri la ricorrenza*

$$I_0 = \log\left(\frac{6}{5}\right), \tag{1.5}$$

$$I_k = \frac{1}{k} - 5\,I_{k-1} \quad k = 1, 2, \ldots n\,, \tag{1.6}$$

*che in teoria dovrebbe convergere a*

$$I_n = \int_0^1 \frac{x^n}{x+5} dx\,,$$

*mentre che cosa possiamo dire circa la convergenza della ricorrenza (1.6)?*

# Capitolo 2

# Ricerca di zeri di funzioni

Dalla nostra esperienza matematica sin dalla scuola media superiore, dato un polinomo di grado $n$, $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$, sappiamo che esistono delle formule esplicite di calcolo delle sue radici, solo per $n \leq 4$, mentre per $n \geq 5$ non esistono formule generali che ci consentono di determinarne gli zeri *in un numero finito di operazioni*. A maggior ragione questo vale nel caso in cui si vogliano determinare le soluzioni di $f(x) = 0$, per una generica funzione $f$.

Queste considerazioni introduttive ci inducono a dire che la ricerca di soluzioni di $f(x) = 0$ si potrà fare solo con *tecniche di tipo iterativo*.

## 2.1 Ricerca di zeri di funzione

La ricerca di *zeri di funzione* è un problema frequente nel calcolo scientifico. Facciamo un paio di esempi

1. *Dinamica della popolazioni.* Consideriamo il seguente modello `preda-predatore`, che modellizza l'evoluzione di una determinata popolazione di cellule, di batteri, di animali ecc... mediante l'equazione

$$x^+ = \frac{rx^2}{1 + \left(\frac{x}{c}\right)^2}, \quad r > 0, \; c > 0 \tag{2.1}$$

   L'equazione (2.1) dice che la popolazione "successiva" $x^+$ cresce secondo una legge non lineare dipendente dai parametri $r, c$ che indicano le risorse disponibili. Scrivendola nella forma $x^+ = g(x)$ (con ovvio significato), ci si potrebbe chiedere se esiste un valore $x^*$ tale che $x^* = g(x^*)$. Questa è la tipica formulazione del problema di un

metodo iterativo per la ricerca di un `punto fisso` dell'equazione (2.1) corrispondente allo zero della funzione $f(x) = x - g(x)$.

2. *Capitale in un fondo d'investimento.* Sia $C$ il capitale che si investe all'inizio di ogni anno su un fondo d'investimento. Il montante dopo il primo anno è $M_1 = C + Cx = C(1+x)$. Dopo $n$ anni il *montante*, $M_n$, sarà la somma dei montanti ottenuti con *capitalizzazione composta*. Ovvero

$$M_n = C(1+x) + C(1+x)^2 + \cdots + C(1+x)^n = C\sum_{i=1}^{n}(1+x)^i\,. \tag{2.2}$$

con $x$ che indica il tasso fisso d'investimento ($x \in (0,1)$). Se desideriamo calcolare il `tasso medio` $x^*$ di rendita del nostro piano d' investimento, chiamando con $f(x) = M_n - C\sum_{i=1}^{n}(1+x)^i$, ancora una volta il problema consiste nella ricerca dello zero di una funzione $f(x) = 0$.

Passiamo ora ai metodi iterativi per la ricerca di zeri di funzione.

## 2.2 Metodo di bisezione

Sia $f \in \mathcal{C}[a,b]$ t.c. $f(a)f(b) < 0$. Sappiamo allora che esiste almeno uno zero di $f$ in $(a,b)$. Consideriamo per semplicità il caso in cui $(a,b)$ contenga `un solo zero`, diciamo $\alpha$, di $f$ (altrimenti *contrarremo* l'intervallo di studio).

L'algoritmo di calcolo si può descrivere come segue.

**Bisezione**

Inizializzazione: $a_0 = a;\ b_0 = b;\ I_0 = (a_0, b_0)$.
*In pratica siamo al passo* $k = 0$.
*Al passo* $k \geq 1$*, determiniamo l'intervallo* $I_k = \frac{1}{2}I_{k-1}$ *come segue.*
1. Calcolo $x_{k-1} = \frac{a_{k-1}+b_{k-1}}{2}$.
2. Se $f(x_{k-1}) = 0$ allora $\alpha = x_{k-1}$; {`abbiamo trovato la radice!`},
   2.1 Altrimenti se $f(a_{k-1})f(x_{k-1}) < 0$ allora $a_k = a_{k-1},\ b_k = x_{k-1}$
   2.2 Altrimenti se $f(a_{k-1})f(x_{k-1}) > 0$ allora $a_k = x_{k-1},\ b_k = b_{k-1}$
   Fine test 2.
3. $x_k = \frac{a_k+b_k}{2},\ \ k = k+1$.
Ritorna al test 2.

Con questo metodo generiamo una successione $\{x_k\}$ che converge verso $\alpha$, poiché essendo $|I_k| = \frac{1}{2^k}|I_0|$, l'errore $e_k$ al passo $k$ verificherà la maggiorazione

$$|e_k| = |x_k - \alpha| < \frac{1}{2}|I_k| = \frac{1}{2^{k+1}}|b-a|\,.$$

Chiedendo poi che $|e_k| < \epsilon$, potremo determinare *a priori* il numero minimo di iterazioni per ridurre l'errore a meno di $\epsilon$

$$k_{min} > \log_2 \left( \frac{|b-a|}{\epsilon} \right) - 1 \, . \tag{2.3}$$

La funzion Matlab/Octave, `bisezione.m` in Appendice C, implementa il metodo di bisezione (richiede la definizione della funzione `fun` di cui si cerca lo zero).

**Osservazioni**.

- Il metodo non garantisce una riduzione progressiva dell'errore ma solo un dimezzamento dell'ampiezza dell'intervallo dove sta $\alpha$.
- Il metodo non tiene conto del reale andamento della funzione $f$ su $I_0 = [a, b]$. Se, ad esempio, $I_0$ è simmetrico rispetto $\alpha$, basterebbe un solo passo per determinare la radice. Invece, l'algoritmo nella sua formulazione originale, non è in grado di verificarlo.
- Infine, se $f$ è una retta il metodo richiederà più di un passo per trovare la radice. Vedremo che, ad esempio con il metodo di Newton (cfr. Sezione 2.4), nel caso in cui la funzione sia una retta lo zero si determinerà, come giusto che sia, con una sola iterazione.

## 2.3 Iterazione di punto fisso

L'idea del metodo è di *trasformare* il problema originale, che consiste nel cercare gli zeri di $f$ risolvendo $f(x) = 0$, in un *problema di punto fisso* $x = g(x)$ la cui soluzione è la stessa del problema originale.

1. Il primo passo è la *trasformazione* di $f(x) = 0$ in un problema di punto fisso $x = g(x)$, con $g$ derivabile in $I_\alpha$ e t.c. $\alpha = g(\alpha)$ se e solo se $f(\alpha) = 0$.
2. Dato un valore iniziale $x_0$ costruiamo il metodo iterativo $x_{k+1} = g(x_k), \;\; k = 0, 1, \ldots$ che genererà la successione $\{x_k\}$ che convergerà verso un punto $\xi = \alpha$.

Per inciso, la funzione $g(x)$ viene detta `funzione d'iterazione` del metodo iterativo.

La trasformazione di $f(x) = 0$ in $x = g(x)$ non è unica. Infatti, se consideriamo $x^4 - 3 = 0$, la possiamo trasformare in $x = x^4 + x - 3$, oppure in $x = \frac{3+5x-x^4}{5}$ o ancora in $x = \sqrt[3]{\frac{3}{x}}$. In generale esistono *infiniti* modi di ottenere una formulazione di punto fisso.

L'altro problema, una volta ottenuta la forma $x = g(x)$, è di chiederci se tutte le funzioni d'iterazione $g(x)$ vanno ugualmente bene. La risposta è negativa.

**Proposizione 1.** *Se $g(x)$ è derivabile in $I_\alpha$ ed esiste un numero $\mu < 1$ t.c.*

$$|g'(x)| \le \mu, \ \ \forall \ x \in I_\alpha \, ; \tag{2.4}$$

*allora $g(x)$ ha un* `unico` *punto fisso $\alpha$. Inoltre la successione generata dal metodo $x_{k+1} = g(x_k)$ converge ad $\alpha$ per ogni scelta di $x_0 \in I_\alpha$. Infine si ha*

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|} = g'(\alpha) \, . \tag{2.5}$$

Da (2.5) deduciamo che le iterazioni di punto fisso convergono almeno linearmente. Infatti per $k > \bar{k}$, $\bar{k}$ sufficientemente grande, l'errore $e_{k+1} = x_{k+1} - \alpha$ ha lo stesso comportamento di quello al passo $k$ a meno di una costante $|g'(\alpha)| \le \mu < 1$.

**Definizione 5.** *Sia $\{x_k\}$ una successione convergente a $\xi$. Consideriamo l'errore assoluto in modulo al passo k, $|e_k| = |x_k - \xi|$. Se esiste un reale $p \ge 1$ e una costante reale positiva $\gamma < +\infty$ t.c.*

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^p} = \lim_{k\to\infty} \frac{|x_{k+1} - \xi|}{|x_k - \xi|^p} = \gamma \, , \tag{2.6}$$

*allora la successione $\{x_k\}$ ha ordine di convergenza $p$.*

Se $p = 1$ e $0 < \gamma < 1$ parleremo di convergenza `lineare`. Se $p = 1$ e $\gamma = 1$ parleremo di convergenza `sublineare`. Nel caso in cui $1 < p < 2$ si dice che la convergenza è `superlineare`. Se $p = 2$ parleremo di convergenza `quadratica`; se $p = 3$ di convergenza `cubica` e così via.

Come conseguenza della precedente definizione, il metodo di iterazione funzionale $x_{k+1} = g(x_k)$ ha ordine di convergenza $p$ se vale la (2.6).

Un *modo alternativo*, ma pratico, di calcolare l'ordine di convergenza $p$, facilmente derivabile dalla definizione (2.6), è il seguente. Osservando che

$$\lim_{k\to\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^p} = \gamma \neq 0$$

si ricava

$$p = \lim_{k\to\infty} \frac{\log|x_{k+1} - \alpha| - \log\gamma}{\log|x_k - \alpha|} \approx \lim_{k\to\infty} \frac{\log|x_{k+1} - \alpha|}{\log|x_k - \alpha|} \, . \tag{2.7}$$

Naturalmente se, come spesso accade, non si conosce la radice $\alpha$, si puoò considerare, in (2.7) invece di $\alpha$, una sua approssimazione $x_m$ con $m$ più grande di $k + 1$.

Esempio 9. Consideriamo la funzione d'iterazione $g(x) = x(2 - qx), \ q > 0$.

(a) Quali sono i punti fissi di $g(x)$.

(b) Per il metodo $x_{k+1} = g(x_k),\ k \geq 0$, determinare l'intervallo $I_\alpha$ di convergenza della radice positiva $\alpha$.

(c) Calcolare l'ordine di convergenza del metodo iterativo di cui al punto precedente.

**Soluzione**.

(a) Risolvendo $x = x(2 - qx)$ si ottengono le soluzioni $x_1 = 0$ e $x_2 = 1/q > 0$.

(b) L'intervallo di convergenza $I_\alpha$ con $\alpha = 1/q$ si ottiene chiedendo che $|g'(1/q)| < 1$. Ora risolvendo $|g'(x)| < 1$, ovvero $|2(1 - qx)| < 1$, si ha

$$\frac{1}{2q} < x < \frac{3}{2q}\ .$$

Questo intervallo contiene la radice $\alpha = 1/q$ e inoltre $g'(1/q) = 0 < 1$ e quindi, come richiesto dalla Proposizione 1, il metodo converge alla radice positiva per $x \in \left(\frac{1}{2q}, \frac{3}{2q}\right)$.

(c) Calcoliamo l'ordine di convergenza verificando per quali $p$ il limite

$$\lim_{k\to\infty} \frac{|x_k(2 - q\,x_k) - \alpha|}{|x_k - \alpha|^p}$$

risulta finito. Per $p = 1$ non è finito perchè il numeratore si comporta come $x^2$ e il denomiatore come $x$. Invece per $p = 2$, numeratore e denominatore si comportano come $x^2$ e quindi il limite sarà finito. Pertanto il metodo converge con ordine 2.

$\diamond\diamond\diamond$

L'esempio appena svolto ci consente di specializzare il concetto di ordine di convergenza di un metodo iterativo.

**Definizione 6.** *Se la funzione d'iterazione è derivabile almeno p volte con continuità in $I_\alpha$, con $\alpha$ un punto fisso semplice di $g(x)$ per cui*

$$g'(\alpha) = g''(\alpha) = \cdots = g^{(p-1)}(\alpha) = 0,\ \ g^{(p)}(\alpha) \neq 0\ ,$$

*allora il metodo d'iterazione ha ordine di convergenza p.*

Tornando all'esempio precedente punto (c), notiamo che $g'(1/q) = 0$ mentre $g''(1/q) = -2q \neq 0$ che come dimostrato ha ordine di convergenza quadratico.

Naturalmente, questo ha senso se conosciamo la radice $\alpha$.

`Test di arresto`

1. *Test sulla differenza tra due iterate successive.* Il metodo iterativo continuerà la ricerca della soluzione finchè $|x_{k+1} - x_k| < \epsilon$. Infatti
$$x_{k+1} - \alpha = g(x_k) - g(\alpha) = g'(\xi_k)(x_k - \alpha), \;\; \xi_k \in [\alpha, x_k] \; . \tag{2.8}$$
Essendo $x_{k+1} - \alpha = x_{k+1} - x_k + x_k - \alpha$ otteniamo
$$x_k - \alpha = \frac{1}{1 - g'(\xi_k)}(x_k - x_{k+1}) \; .$$
Pertanto, se $g'(x) \approx 0, \; x \in I_\alpha$, in particolare per $x = \xi_k$, allora l'errore viene stimato abbastanza accuratamente dalla differenze delle iterate successive. Invece, se $g'(x) \approx 1$ il fattore $1/(1 - g'(\xi_k)) \to \infty$, pertanto la stima non sarà una buona stima.
2. *Test sulla differenza "relativa" tra due iterate successive.* Il test che faremo ora è
$$|x_{k+1} - x_k| < \epsilon |x_{k+1}| \; .$$
3. *Test sul valore della funzione.* Il test consiste nel verificare se $|f(x_k)| < \epsilon$. Purtroppo questo test non funziona quando la funzione è `piatta` su $I_\alpha$, facendo fermare le iterazioni troppo lontano dal valore della soluzione. Un *esempio*: la funzione $f(x) = (x^{10} - 10)/x$ nell'intorno sinistro della radice positiva $\alpha \approx 1.26$ è molto piatta e usando il test in esame partendo da $x_0 \in I_\alpha = [1, 2]$ usando anche una tolleranza alta come $\epsilon = 1.e - 2$, ci arresteremo dopo migliaia di iterazioni.

L'esempio proposto al punto 3, ci suggerisce le seguenti considerazioni.

- Nei test di arresto è necessario inserire anche un controllo sul numero massimo di iterazioni, $k \leq kmax$.
- Il test che ci darà "maggiore sicurezza" è quindi la combinazione del test sull'errore relativo e il controllo sul numero di passi. Pertanto il metodo iterativo continuerà a cercare la radice finchè
$$(|x_{k+1} - x_k| \geq \epsilon |x_{k+1}|) \;\; \& \;\; (k \leq kmax) \; . \tag{2.9}$$
altrimenti se una delle due condizioni non sarà verificata ci si arresterà.

La funzione Matlab/Octave `MetIterazioneFunz.m`, in Appendice C, implementa un metodo di iterazione funzionale la cui funzione d'iterazione descritta in un altro M-file `g`, richiede in input il valore iniziale `x0`, la tolleranza `tol` e il numero massimo d'iterazioni `kmax`, e restituisce la soluzione in `x0`, `niter`, numero di iterazioni fatte e un `flag` per la convergenza. Se c'è convergenza `flag=1` altrimenti `flag=0`.

*Esercizio.* Trovare un metodo di iterazione funzionale convergente alla radice di $x^{10} = 10$.

## 2.4 Il metodo di Newton o delle tangenti

Supponiamo che $f$ sia derivabile su $[a, b]$. Pertanto possiamo considerare l'equazione della tangente di $f$ in $x_k$

$$y(x) = f(x_k) + (x - x_k)f'(x_k) \ . \tag{2.10}$$

Come punto $x_{k+1}$ prendiamo il punto in cui la retta tangente interseca l'asse delle ascisse. In pratica dobbiamo risolvere $y(x) = 0$.

Imponendo questa condizione in (2.10), otteniamo la *formula del metodo di Newton*

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \ , \tag{2.11}$$

purchè $f'(x_k) \neq 0, \ \ k \geq 0$.

Facciamo ora un paio di osservazioni.

1. Il metodo di Newton consiste nel sostituire localmente $f(x)$ con la retta tangente. Infatti
$$f(x_{k+1}) = f(x_k) + (x_{k+1} - x_k)f'(x_k) + \mathcal{O}((x_{k+1} - x_k)^2) \ ,$$
da cui, imponendo che $f(x_{k+1}) = 0$ e trascurando i termini di ordine superiore al primo, otteniamo la (2.11). Questo ci dice che la (2.11) è un modo per approssimare $f$ in $x_{k+1}$.

2. Se $f(x) = a_0 + a_1 x$ ($f$ è una retta), allora il metodo di Newton converge in una sola iterazione. Infatti
$$x_1 = x_0 - \frac{a_0 + a_1 x}{a_1} = -\frac{a_0}{a_1} \ .$$

$\diamond\diamond$

Facciamo ora vedere che se $x_0$ è preso *" sufficientemente"* vicino alla radice $\alpha$, con $f'(\alpha) \neq 0$ (ovvero $\alpha$ radice semplice), allora il metodo converge almeno quadraticamente e si ha

$$\lim_{k\to\infty} \frac{x_{k+1} - \alpha}{(x_k - \alpha)^2} = \frac{f''(\alpha)}{2f'(\alpha)} \ , \tag{2.12}$$

da cui, se $f''(\alpha) \neq 0$, allora il metodo converge quadraticamente altrimenti con ordine maggiore di due.

Dimostriamo la (2.12).

$$\begin{aligned} 0 = f(\alpha) &= f(x_k) + f'(x_k)(\alpha - x_k) + \frac{(\alpha - x_k)^2}{2} f''(\xi), \;\; \xi \in (x_k, \alpha) \\ &= \frac{f(x_k)}{f'(x_k)} + \alpha - x_k + \frac{(\alpha - x_k)^2}{2f'(x_k)} f''(\xi) \\ &= x_k - x_{k+1} + \alpha - x_k + \frac{(\alpha - x_k)^2}{2f'(x_k)} f''(\xi) \\ &= \alpha - x_{k+1} + (\alpha - x_k)^2 \frac{f''(\xi)}{2f'(x_k)} \end{aligned}$$

si conclude dividendo per $(x_k - \alpha)^2$, portando a primo membro e passando al limite. □

Il seguente teorema ci da delle condizioni per la *convergenza globale* del metodo di Newton.

**Teorema 1.** *Sia $f \in \mathcal{C}^2[a,b]$ con [a,b] chiuso e limitato, inoltre*

1. $f(a)f(b) < 0$
2. $f'(x) \neq 0, \;\; x \in [a,b]$
3. $f''(x) \geq 0$ *oppure* $f''(x) \leq 0, \;\; \forall x \in [a,b]$
4. $\left|\frac{f(a)}{f'(a)}\right| < b - a$ *e* $\left|\frac{f(b)}{f'(b)}\right| < b - a.$

*Allora il metodo di Newton converge all'* `unica` *soluzione $\alpha \in [a,b]$ per ogni $x_0 \in [a,b]$.*

*Osservazione.* L'ultima ipotesi del Teorema ci assicura che la tangente agli estremi $a$ e $b$ interseca l'asse $x$ all'interno di $[a,b]$.

**Dim.** Supponiamo, come visualizzato in figura 2.1, che $f' > 0$, $f'' \leq 0$ e $f(a) < 0$, $f(b) > 0$ (ovvero nell'ipotesi di esistenza di un unico zero $\alpha$ in $[a,b]$).

Sia $a \leq x_0 < \alpha$ e, ovviamente, $f(x_0) \leq 0 = f(\alpha)$. Allora $x_1 = x_0 - f(x_0)/f'(x_0) \geq x_0$. Proviamo per induzione che $x_k \leq \alpha$ e $x_{k+1} \geq x_k$.

Per $k = 0$ è vera. Sia vera per $k$ e proviamola per $k+1$.

$$-f(x_k) = f(\alpha) - f(x_k) = (\alpha - x_k) f'(\xi_k), \;\; x_k \leq \xi_k \leq \alpha \,.$$

Ma, $f''(x) \leq 0$, che implica che $f'$ è decrescente. Quindi $f'(\xi_k) \leq f'(x_k)$. Allora,

$$\begin{aligned} -f(x_k) &\leq (\alpha - x_k) f'(x_k) \\ x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} \leq x_k + (\alpha - x_k) = \alpha \,. \end{aligned}$$

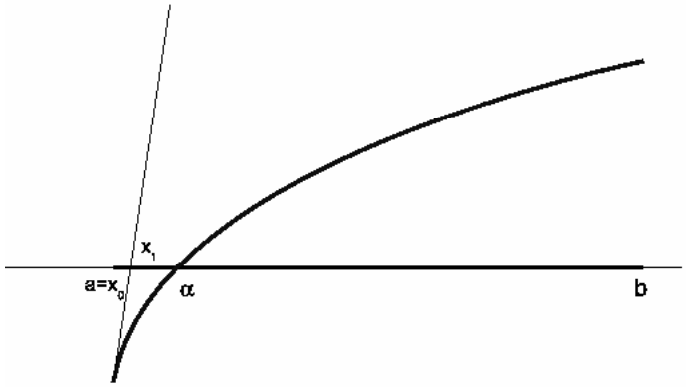


**Figura** 2.1: Interpretazione geometrica della condizione 4 del Teorema 1 nell'ipotesi di funzione è concava in $[a, b]$

Segue che $f(x_{k+1}) \leq f(\alpha) = 0$ e anche che $x_{k+2} \geq x_{k+1}$ come richiesto.

In conclusione, la successione $\{x_k\}$ è monotona e limitata superiormente e quindi convergente: $\lim_{k\to\infty} x_k = \alpha$ . □

Se $\alpha$ è `zero multiplo`, con molteplicità $m > 1$ il metodo di Newton converge linearmente. Vediamolo su un semplice esempio.

ESEMPIO 10. $f(x) = x^2$. Il metodo di Newton costruisce la successione

$$x_{k+1} = x_k - \frac{x_k^2}{2x_k} = \frac{x_k}{2} .$$

L'errore corrispondente soddisfa la successione $e_{k+1} = \frac{e_k}{2}$ che ci dice appunto che il metodo converge linearmente.

Se si considerasse la successione

$$x_{k+1} = x_k - 2\frac{x_k^2}{2x_k} = 0$$

il metodo converge immediatamente alla radice doppia $\alpha = 0$.

◇

L'esempio ci suggerisce come modificare il metodo di Newton affinchè sia mantenuta la convergenza quadratica anche in presenza di zeri con molteplicità $m > 1$.

$$x_{k+1} = x_k - m\,\frac{f(x_k)}{f'(x_k)},\quad f'(x_k)\neq 0,\ k\geq 0\,. \tag{2.13}$$

La successione generata con l'iterazione (2.13) converge quadraticamente alla radice multipla $\alpha$ alla luce della seguente osservazione: *il metodo di Newton è un metodo di iterazione funzionale con funzione d'iterazione* $g(x) = x - \frac{f(x)}{f'(x)}$.

Facciamo vedere che, nel caso in cui la radice $\alpha$ ha molteplicità $m$, per mantenere l'ordine di convergenza quadratico, dobbiamo considerare la funzione d'iterazione

$$g(x) = x - m\frac{f(x)}{f'(x)}\,. \tag{2.14}$$

Infatti, poichè possiamo scrivere $f(x) = (x-\alpha)^m h(x)$ con $h^{(p)}(\alpha)\neq 0,\ p=0,\ldots,m$ e

$$\begin{aligned} g(x) &= x - \frac{(x-\alpha)\,h(x)}{mh(x)+(x-\alpha)h'(x)} \\ g'(x) &= 1 - \frac{h(x)}{mh(x)+(x-\alpha)h'(x)} - (x-\alpha)\frac{d}{dx}\psi(x) \end{aligned}$$

dove $\psi(x) = \frac{h(x)}{mh(x)+(x-\alpha)h'(x)}$. Pertanto $g'(\alpha) = 1 - 1/m \neq 0$ se $m > 1$. È facile a questo punto verificare che se prendiamo $g(x) = x - m\,f(x)/f'(x)$, come in (2.14), allora $g'(\alpha) = 0$ che ci garantisce ancora convergenza almeno quadratica del metodo di Newton anche per zeri con multeplicità $m > 1$.

*Se non conosciamo la molteplicità della radice*, considereremo invece di $f(x)$ la funzione $\phi(x) = f(x)/f'(x)$ e applicheremo il metodo di Newton a questa funzione costruendo la successione

$$x_{k+1} = x_k - \frac{\phi(x_k)}{\phi'(x_k)}\,.$$

L'unico inconveniente di questa tecnica è che si deve calcolare la derivata seconda della funzione $f$. Alternativamente, si può stimare il valore della molteplicità con una successione

$$m_k = \frac{x_{k-1}-x_{k-2}}{2x_{k-1}-x_k-x_{k-2}} \tag{2.15}$$

come descritto in [20, §6.2.2]. Infatti, visto che la successione $\{x_k\}$ converge (linearmente) alla radice $\alpha$ allora

$$\lim_{k\to\infty}\frac{x_k-x_{k-1}}{x_{k-1}-x_{k-2}} = \lim_{k\to\infty}\frac{g(x_{k-1})-g(x_{k-2})}{x_{k-1}-x_{k-2}} = g'(\alpha) = 1-\frac{1}{m}\,,$$

da cui

$$\lim_{k\to\infty}\frac{1}{1-\frac{x_k-x_{k-1}}{x_{k-1}-x_{k-2}}} = m\,.$$

Vediamo ora un paio di esempi (didattici ma importanti).

1. $f(x) = x^2 - q,\ x > 0,\ q \in \mathbb{R}_+$. Il problema ha soluzione $x = \sqrt{q}$. La successione del metodo di Newton è
$$x_{k+1} = \frac{1}{2}(x_k + \frac{q}{x_k})\,,$$
che altro non è che il `metodo babilonese o di Erone` che calcola $\sqrt{q}$ usando le operazioni elementari. Poiché $f' > 0,\ f'' > 0$ per $x > 0$, allora per il Teorema 1, per ogni $0 < a < \sqrt{q} < b$ la successione converge a $\sqrt{q}$.
Nel caso $f(x) = x^n - q,\ q > 0,\ n > 0$,
$$x_{k+1} = x_k(1 - \frac{1}{n}) + \frac{q}{n}x_k^{1-n}$$
consente di calcolare la radice $n$-esima del numero reale positivo $q$.

2. $f(x) = \frac{1}{x} - c$. Il problema equivale quindi a calcolare l'inverso di $c$. Supponiamo, per semplicità che $c > 0$. La successione del metodo di Newton è
$$x_{k+1} = x_k(2 - cx_k)$$
che consente di calcolare il reciproco senza divisioni! Ora, per applicare il Teorema 1, osservo che essendo $f' < 0$ e $f'' > 0$ (ricorda che $x > 0$) dovrò trovare $c$, il con $a < 1/c < b$, tale che
$$\frac{f(b)}{f'(b)} = b(bc - 1) < b - a \iff \frac{1 - \sqrt{1 - ac}}{c} < b < \frac{1 + \sqrt{1 - ac}}{c}$$
Pertanto, se $a > 0$ il metodo di Newton converge pur di prendere $\dfrac{1}{2c} < x_0 < \dfrac{3}{2c}$.

Concludiamo con un uleriore esempio.

Esempio 11. Data la funzione
$$f_\alpha(x) = \frac{\sin(\alpha x)}{\alpha x + 2}\log(\alpha x),\ \ \alpha \neq 0\,,$$

(a) dire quali sono gli zeri di $f_\alpha(x)$ risolvendo analiticamente $f_\alpha(x) = 0$;

(b) per $\alpha = 2$, si calcoli lo zero $x^* = 1/\alpha$ mediante il metodo di Newton a meno di $tol = 1.e - 6$.

Anzitutto la funzione è definita per $x \neq -2/\alpha$. Ma la condizione sull'esistenza del logaritmo, richiede che $\alpha x > 0$ che implica che $\alpha$ e $x$ siano concordi in segno. Pertanto, il suo campo di esistenza è $\mathbb{R} \setminus \{-2/\alpha\}$. Gli zeri si ottengono dalle equazioni e disequazioni
$$\begin{aligned} \sin(\alpha x) &= 0\,, \\ \log(\alpha x) &= 0\,, \\ \alpha\, x &> 0\,. \end{aligned}$$

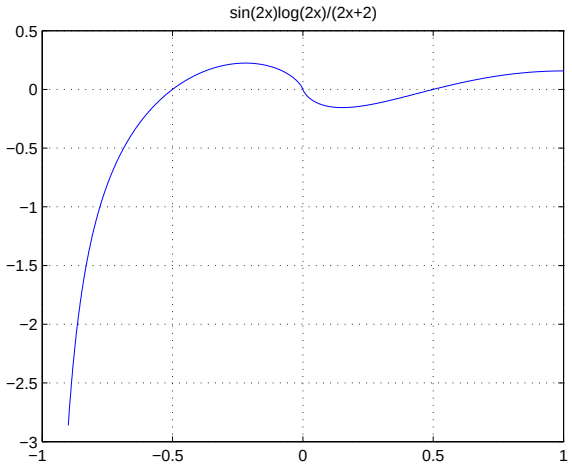


**Figura** 2.2: La funzione dell'Esempio 11 in $[0.9, 1]$ con $\alpha = 2$.

che hanno soluzioni $x = \frac{k\pi}{\alpha}$, $k \in \mathbb{Z}$, $x = 1/\alpha$ e $\alpha\, x > 0$ . In $x = 0$ la funzione è definita per continuità e vale 0.

Ad esempio, per $\alpha = 2$, lo zero richiesto è $x^* = 1/2$. Usando il metodo di Newton, sapendo che $f'_\alpha(x) = \dfrac{\sin(\alpha x)}{x(\alpha x + 2)} + \alpha \left[\dfrac{\cos(\alpha x)(\alpha x + 2) - \sin(\alpha x)}{(\alpha x + 2)^2}\right]$, con il codice seguente Matlab/Octave:

```
 kmax=100; tol=1.e-6;
 x0=3/(2*a);
 iter(1)=x0;
 [y,yd]=fun1(x0,a);
 x1=x0-y/yd;
 k=1;
 iter(k+1)=x1;
while abs(x1-x0)> tol*abs(x1) & k <=kmax
    x0=x1;
    [y,yd]=fun1(x0,a);
    x1=x0-y/yd;
    iter(k+1)=x1;
    k=k+1;
end
disp('La soluzione cercata e'' '); x1
%---file che valuta la funzione e la sua derivata  ----
function [y,yd]=fun1(x,a)
Ax=a*x+2; Sx=sin(a*x);
y=Sx./Ax.*log(a*x);
```

```
yd=Sx./(Ax.*x)+a*(cos(a*x).*Ax-Sx)./(Ax.^2);
return
```

in 12 iterazioni si calcola la soluzione richiesta.

$\diamond\diamond$

### 2.4.1 Varianti del metodo di Newton

Descriviamo brevemente alcune varianti del metodo di Newton note in letteratura con altri nomi.

#### Metodo delle corde

Consiste nel considerare costante, uguale ad un certo valore $c$, la derivata prima della funzione $f$. Si ottiene pertanto il `metodo delle corde`

$$x_{k+1} = x_k - \frac{f(x_k)}{c}, \quad c \in \mathbb{R}\backslash\{0\}. \tag{2.16}$$

Per la ricerca del *valore ottimale* per $c$, si deve tener conto del fatto che il metodo è un metodo d'iterazione funzionale con funzione d'iterazione $g(x) = x - f(x)/c$. Pertanto $c$ si sceglie cosicché

$$|g'(x)| = \left|1 - \frac{f'(x)}{c}\right| < 1\,,$$

in un intorno $I_\alpha = [\alpha - \delta, \alpha + \delta]$ della soluzione $\alpha$. Pertanto, per la convergenza del metodo delle corde dovremo verificare le seguenti condizioni:

$$\begin{aligned} f'(x) &\neq 0, \quad x \in I_\alpha, \\ 0 < f'(x)/c &< 2\,. \end{aligned}$$

Dalla seconda condizione, indicando con $M = \max_{x\in I_\alpha} |f'(x)|$ si deduce che per la convergenza dobbiamo richiedere che $|c| > M/2$ e anche che $c\, f'(x) > 0$.

Se $c \neq f'(\alpha)$ allora il metodo ha convergenza lineare, quando $c = f'(\alpha)$ il metodo è almeno del primo ordine.

**Metodo delle secanti**

L'idea è quello di approssimare $f'(x_k)$, che appare nel metodo di Newton, con il rapporto incrementale $\dfrac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$. Si ottiene

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k)-f(x_{k-1})}, \quad k = 1, 2, ... \tag{2.17}$$

con $f(x_{k-1}) \neq f(x_k)$. Pertanto il metodo richiede la conoscenza di due valori iniziali, $x_0,\ x_1$. Al passo $k$, il nuovo valore $x_{k+1}$ è l'intersezione della secante, ovvero la retta per i punti $(x_{k-1}, f(x_{k-1}))$ e $(x_k, f(x_k))$, con l'asse delle ascisse.

Il metodo delle secanti converge, sotto le stesse ipotesi del metodo di Newton, con ordine di convergenza

$$p = \frac{1+\sqrt{5}}{2} \approx 1.618\,,$$

che equivale ad una convergenza `superlineare`. Ma, l'importanza del metodo delle secanti stà principalmente nel costo computazionale: *il metodo richiede* `solo` *il calcolo di* $f(x_k)$ mentre il metodo di Newton richiede i valori di $f(x_k)$ e di $f'(x_k)$. Nel caso di funzioni la cui espressione è "complicata", il calcolo della derivata può essere costoso dal punto di vista della complessità. Pertanto il metodo delle secanti, pur avendo una convergenza superlineare, rappresenta sempre una valida alternativa al metodo di Newton.

Nel valutare, se usare il metodo delle secanti o di Newton, si dovrebbe considerare la loro `efficienza computazionale` che indica se è più costoso calcolare la derivata o il rapporto incrementale senza tralasciare il fatto che il calcolo della derivata di una funzione è comunque un problema mal-condizionato.

Osserviamo che in [2] viene chiamato metodo delle secanti il metodo iterativo

$$x_{k+1} = x_k - f(x_k)\frac{x_k - c}{f(x_k)-f(c)}, \quad k = 1, 2, ... \tag{2.18}$$

con $c \in [a, b]$, che corrisponde ad usare una secante sempre con la stessa pendenza. In questo caso, la convergenza è di tipo lineare. Se $c$ è scelto cosicché $f(c)/(c-\alpha)$ ha lo stesso segno di $f'(\alpha)$ ed inoltre

$$\left|\frac{f(c)}{c-\alpha}\right| > \frac{1}{2}|f'(\alpha)|$$

allora la corrispondente funzione d'iterazione è tale che $|g'(x)| < 1$ e quindi il metodo converge.

**Il metodo di Steffensen**

Il metodo costruisce la sequenza

$$x_{k+1} = x_k - \frac{f(x_k)}{g(x_k)}, \tag{2.19}$$

$$g(x_k) = \frac{f(x_k + f(x_k)) - f(x_k)}{f(x_k)}. \tag{2.20}$$

Posto $\beta_k = f(x_k)$, si ha

$$g(x_k) = \frac{f(x_k + \beta_k) - f(x_k)}{f(x_k)} = f'(x_k)\left(1 - \frac{1}{2}h_k f''(x_k) + \mathcal{O}(\beta_k^2)\right)$$

con $h_k = -f(x_k)/f'(x_k)$ che è la correzione di Newton.

Osservando che per la funzione $s(x) = 1/(1-x)$ si può scrivere come $s(x) = 1+x+\mathcal{O}(x^2)$, pertanto la (2.19) diventa

$$x_{k+1} = x_k + h_k(1 + \frac{h_k}{2}f''(x_k) + \mathcal{O}(\beta_k^2)) . \tag{2.21}$$

Da cui, per l'errore $e_k = x_k - \alpha$, osservando che

$$h_k = -e_k + \frac{1}{2}e_k^2 \frac{f''(\xi)}{f'(x_k)}$$

(si ottiene dal fatto che $h_k = (x_k - \alpha) + \dfrac{(x_k-\alpha)^2}{2}\dfrac{f''(\xi_k)}{f'(x_k)}$)

otteniamo

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \frac{f''(\alpha)}{2f'(\alpha)}(1 + f'(\alpha)) .$$

In conclusione il metodo di Steffensen è un metodo di ordine 2.

## 2.5 Accelerazione di Aitken

Il metodo consente di accelerare una sequenza ottenuta a partire da successioni di punto fisso $x_{k+1} = g(x_k)$, $k \geq 0$.

Se $\{x_k\}$ converge linearmente allo zero $\alpha$, allora possiamo dire che esiste un $\eta$ (da determinarsi) tale che

$$g(x_k) - \alpha = \eta(x_k - \alpha) . \tag{2.22}$$

Il metodo si propone di definire una "nuova" successione che migliori la successione ottenuta con il metodo di partenza. Dalla (2.22) otteniamo

$$\alpha = \frac{g(x_k) - \eta x_k}{1-\eta} = \frac{g(x_k) - \eta x_k + x_k - x_k}{1-\eta}$$

ovvero

$$\alpha = x_k + \frac{g(x_k) - x_k}{1-\eta} \, . \tag{2.23}$$

Come possiamo determinare $\eta$? Lo approssimiamo con la successione

$$\eta_k = \frac{g(g(x_k)) - g(x_k)}{g(x_k) - x_k} \, . \tag{2.24}$$

**Lemma 1.** *Se la successione $x_{k+1} = g(x_k)$ converge ad $\alpha$ allora*

$$\lim_{k \to +\infty} \eta_k = g'(\alpha) \, .$$

**Dim.** Osserviamo che $x_{k+1} = g(x_k)$ e $x_{k+2} = g(g(x_k))$. Da (2.24)

$$\begin{aligned} \eta_k &= \frac{x_{k+2} - x_{k+1}}{x_{k+1} - x_k} = \frac{x_{k+2} - \alpha - (x_{k+1} - \alpha)}{x_{k+1} - \alpha - (x_k - \alpha)} = \\ &= \frac{\frac{x_{k+2} - \alpha}{x_{k+1} - \alpha} - 1}{1 - \frac{x_k - \alpha}{x_{k+1} - \alpha}} \, . \end{aligned}$$

Passando al limite, ricordando che per ipotesi la successione converge ovvero che $\lim_{k \to +\infty} \dfrac{x_{k+1} - \alpha}{x_k - \alpha} = g'(\alpha)$, otteniamo l'asserto

$$\lim_{k \to +\infty} \eta_k = \frac{g'(\alpha) - 1}{1 - \frac{1}{g'(\alpha)}} = g'(\alpha) \, .$$

In definitiva $\{\eta_k\}$ approssima $\eta$. $\square$

Usando (2.23) e (2.24) otteniamo la "nuova successione"

$$\hat{x}_{k+1} = x_k - \frac{(g(x_k) - x_k)^2}{g(g(x_k)) - 2g(x_k) + x_k}, \;\; k \geq 0 \tag{2.25}$$

detta `formula di estrapolazione di Aitken` o anche `metodo di Steffensen`. La (2.25) si può considerare una iterazione di punto fisso con funzione d'iterazione

$$g_\Delta(x) = \frac{x\, g(g(x)) - (g(x))^2}{g(g(x)) - 2g(x) + x} \, .$$

$\diamond$

**Osservazione**. Il $\Delta$ a pedice nella $g_\Delta$ è dovuto alla seguente osservazione. Osserviamo che la successione di Aitken si può riscrivere come

$$\hat{x}_{k+1} = x_k - \frac{(x_{k+1} - x_k)^2}{x_{k+2} - 2x_{k+1} + x_k}\,, \tag{2.26}$$

dove appare evidente la presenza dell'`operatore differenze in avanti`, $\Delta$. $\Delta$ è un operatore lineare che si definisce come

$$\Delta\, x = (x + h) - x, \;\; h > 0$$

Pertanto $\Delta\, x_k = x_{k+1} - x_k$, $\;\Delta^2\, x_k = \Delta(\Delta\, x_k) = \Delta x_{k+1} - \Delta\, x_k = x_{k+2} - 2x_{k+1} + x_k$. In definitiva la successione di Aitken (2.26), usando l'operatore $\Delta$, diventa

$$\hat{x}_{k+1} = x_k - \frac{(\Delta\, x_k)^2}{\Delta^2\, x_k}\,. \tag{2.27}$$

Talvolta, per indicare il metodo di accelerazione di Aitken, si usa la notazione $\Delta^2$ `di Aitken`.

$\diamond$

Tornando alla $g_\Delta(x)$, notiamo che è indeterminata per $x = \alpha$. Infatti, ricordando che $g(\alpha) = \alpha$ e $g(g(\alpha)) = \alpha$ otteniamo $g_\Delta(\alpha) = \frac{\alpha^2 - \alpha^2}{\alpha - 2\alpha + \alpha} = \frac{0}{0}$. Se $g$ è derivabile e $g'(\alpha) \neq 1$ allora applicando de l' Hôpital $\lim_{x\to\alpha} g_\Delta(x) = \alpha$. Pertanto, in $x = \alpha$, $g_\Delta(x)$ è estendibile per continuità e $g_\Delta(\alpha) = \alpha$.

Se $g(x) = x - f(x)$ allora $g'(\alpha) = 1$ se e solo se $\alpha$ ha molteplicità 2. Anche per questa particolare $g$, si verifica che $g_\Delta(\alpha) = \alpha$ ovvero ha gli stessi punti fissi di $g$. Possiamo quindi considerare l'iterazione di punto fisso $x_{k+1} = g(x_k)$, $g(x) = x - f(x)$. Vale il seguente risultato.

**Proposizione 2.** *Sia $g(x) = x - f(x)$ e $\alpha$ radice di $f$. Se $f$ è sufficientemente regolare la successione $x_{k+1} = g(x_k)$ ha le seguenti proprietà.*

*(i) se le iterazioni di punto fisso convergono linearmente ad una radice semplice di $f$ allora $\Delta^2$ di Aitken converge quadraticamente alla stessa radice.*

*(ii) se le iterazioni di punto fisso convergono con ordine $p \geq 2$ ad una radice semplice di $f$ allora $\Delta^2$ di Aitken converge alla stessa radice con ordine $2p - 1$.*

*(iii) se le iterazioni di punto fisso convergono linearmente ad una radice multipla di molteplicità $m \geq 2$ di $f$ allora $\Delta^2$ di Aitken converge linearmente alla stessa radice con fattore asintotico $1 - 1/m$.*

*Inoltre, nel caso $p = 1$ con $\alpha$ radice semplice di $f$, il metodo di Aitken converge anche se le corrispondenti iterazioni di punto fisso non convergono.*

ESEMPIO 12. La funzione $\tan(x) = \frac{3}{2}x - \frac{1}{10}$ ha la radice $\alpha = 0.205921695$. Se la determiniamo con il metodo iterativo $x_{k+1} = 2\dfrac{0.1 + \tan(x_k)}{3}$ partendo da $x_0 = 0$ otteniamo una successione linearmente convergente ad $\alpha$ (infatti $g'(\alpha) \approx 0.45636 < 1$). In Tabella 2.1 facciamo vedere la differente velocità di convergenza usando anche la successione del metodo di accelerazione $\Delta^2$ di Aitken.

| $k$ | $x_k$ | $\hat{x}_k$ (Aitken) |
|---|---|---|
| 0 | 0 | 0 |
| ⋮ | ⋮ | ⋮ |
| 2 | 0.111 | 0.2024 |
| ⋮ | ⋮ | ⋮ |
| 5 | 0.1751 | 0.2053 |

Tabella 2.1: Confonto di una successione di punto fisso e di $\Delta^2$ di Aitken

Una possibile implemetazione del metodo di accelerazione di Aitken, in Matlab/Octave, è descritta nella funzione Aitken.m nell'Appendice C.

## 2.6 Calcolo delle radici di polinomi algebrici

Indicheremo con

$$p_n(x) = \sum_{k=0}^{n} a_k x^k, \quad a_k \in \mathbb{R}$$

un poliomio algebrico di grado $n$. Per la ricerca delle radici reali e/o complesse di $p_n(x)$ ricordiamo anzitutto due risultati utili a comprendere la difficoltà del problema.

- *Regola dei segni di Cartesio.* Dato $p_n(x)$, indichiamo con $s$ il numero di cambiamenti di segno nell'insieme dei coefficienti $\{a_k\}$ e con $p$ il numero delle radici reali positive ognuna contata con la propria molteplicità. Allora $p \le s$ e $s - p$ è un numero pari.
- *Regola di Cauchy.* Tutti gli zeri di $p_n(x)$ sono inclusi nel cerchio $\Omega \subset \mathbb{C}$

$$\Omega = \{z \in \mathbb{C} : \ |z| \le 1 + \gamma\}, \quad \gamma = \max_{0 \le k \le n-1} \left| \frac{a_k}{a_n} \right|$$

Vediamo ora un paio di esempi esplicativi che ci dicono come la regola di Cauchy ci dia una localizzazione troppo approssimativa.

1. Sia $p_3(x) = x^3 - 3x + 2$ (che si può fattorizzare $(x-1)^2(x+2)$). Questo polinomio ha $s = 2$, $p = 2$ quindi la regola di Cartesio vale in quanto $2 \leq 2$ e $2 - 2 = 0$ è pari. Pe Cauchy abbiamo che il cerchio di raggio $1 + \gamma = 1 + 3 = 4$ contiene le radici.

2. Sia $p_6(x) = x^6 - 2x^5 + 5x^4 - 6x^3 + 2x^2 + 8x - 8$ le cui radici sono $\pm 1$, $\pm 2i$, $1 \pm i$. Abbiamo una sola radice positiva: $p = 1$. Il numero dei cambi di segno è $s = 5$. Anche qui le due regole di Cartesio e Cauchy sono ancora vere. In particolare per Cauchy avremo che $\gamma = 8$!

### 2.6.1 Schema di Hörner

Lo schema consente di valutare efficientemente un polinomio in un punto. Partiamo con un esempio esplicativo. Per valutare il polinomio $p_2(x) = a_0 + a_1 x + a_2 x^2$ in un punto $\zeta$ richiederebbe 2 addizioni e 2 moltiplicazioni. Se lo riscrivessimo nella forma equivalente $p_2(x) = a_0 + x(a_1 + a_2 x)$, per valutarlo in $\zeta$ occorerebbero 2 addizioni e 2 moltiplicazioni.

Nel caso generale, la valutazione in $\zeta$ di $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$ richiederebbe $n$ somme e $2n - 1$ moltiplicazioni. Usando la riscrittura

$$p_n(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_n x)))$$

serviranno solo $n$ somme e $n$ moltiplicazioni.

L'algoritmo di Hörner per valutare $p_n(x)$ nel punto $\zeta$ si può così descrivere.

| |
|---|
| $b_n = a_n$ ; |
| for k=n-1:-1:0, |
| $b_k = a_k + b_{k+1}\zeta$ |
| end for |
| Alla fine $b_0 = p_n(\zeta)$. |

Tabella 2.2: Algoritmo di Hörner per la valutazione di un polinomio $p_n(x)$ nel punti $\zeta$.

L'algoritmo di Hörner è anche detto di `divisione sintetica`. Infatti, consideriamo il polinomio

$$q_{n-1}(x;\zeta) = b_1 + b_2 x + \cdots + b_n x^{n-1}$$

i cui coefficienti sono i coefficienti $b_k$ calcolati con l'algoritmo di Hörner e che dipendono da $\zeta$, allora possiamo scrivere

$$p_n(x) = (x - \zeta) q_{n-1}(x;\zeta) + b_0$$

con $b_0$ che è il *resto della divisione* di $p_n(x)$ per $x - \zeta$. Per Ruffini sappiamo che $b_0 = p_n(\zeta)$ e quindi $b_0 = 0$ quando $\zeta$ è una radice di $p_n(x)$. Pertanto, quando $p_n(\zeta) = 0$ possiamo

scrivere

$$p_n(x) = (x - \zeta)q_{n-1}(x; \zeta) \ .$$

Per determinare le rimanenti radici di $p_n(x)$ dobbiamo risolvere l'equazione $q_{n-1}(x; \zeta) = 0$. Per fare questo opereremo per *deflazione* come descriveremo nel prossimo algoritmo che dovremo eseguire per ogni valore di $k$ da $n$ fino a 1 (ovvero `k=n:-1:1`).

**Algoritmo 1.**

(i) trova una radice $\zeta_k$ di $p_k$ con un metodo di ricerca radici (es. Newton);

(ii) calcola il polinomio quoziente $q_{k-1}(x; \zeta_k)$ usando lo schema di Hörner;

(iii) poni $p_{k-1} = q_{k-1}$ e vai a (i).

**Metodo di Newton-Hörner**

È il metodo di Newton associato allo schema di deflazione: calcola la radice $\zeta_k$ di $p_k(x)$. Osservo anzitutto che se $p_n(x) = (x - \zeta)q_{n-1}(x)$ allora

$$p_n'(x) = q_{n-1}(x; \zeta) + (x - \zeta)q_{n-1}'(x; \zeta)$$

Da cui

$$p_n'(\zeta) = q_{n-1}(\zeta; \zeta) \ .$$

Pertanto il metodo di Newton-Hörner per approssimare la j-esima radice $\zeta_j,\ j = 1, \ldots, n$ di $p_n$, consiste, a partire da una approssimazione iniziale $\zeta_j^{(0)}$, nel costruire la successione

$$\zeta_j^{(k+1)} = \zeta_j^{(k)} - \frac{p_n(\zeta_j^{(k)})}{q_{n-1}(\zeta_j^{(k)}; \zeta_j^{(k)})} \ .$$

Poi, ricordando che $p_n(x) = (x - \zeta_j)q_{n-1}(x)$ si sfrutta la deflazione per approssimare uno zero di $q_{n-1}$ finchè determineremo tutte le radici.

## 2.7 Esercizi proposti

ESERCIZIO 9. `(Laboratorio del 2/11/05)`. *Data la funzione* $f(x) = \cosh x + \sin x - \gamma$, *per* $\gamma = 1, 2, 3$ *si individui graficamente un intervallo contenente uno zero* $\xi \geq 0$ *e lo si calcoli con il metodo di bisezione con* $tol = 1.e-10$. *Calcolare anche il numero di iterazioni necessarie sia a priori che a posteriori. Fare anche il grafico dell'errore relativo da cui si evince che la convergenza è lineare.*

ESERCIZIO 10. (Laboratorio del 2/11/05). *Un oggetto si trova fermo su un piano la cui inclinazione varia con velocità costante $\omega$. Dopo t secondi la posizione del questo oggetto è*

$$s(t,\omega) = \frac{g}{2\omega^2}\left(\sinh(\omega t) - \sin(\omega t)\right)$$

*dove $g = 9.81 m/sec^2$ è l'accelerazione di gravità. Supponiamo che il corpo si sia mosso di 1 metro in 1 secondo. Si ricavi il valore corrispondente di $\omega$ con accuratezza $1.e-5$, mediante un metodo di iterazione funzionale convergente! (Sugg: si deve trovare una funzione di iterazione la cui derivata prima risulta in modulo minore di 1 nell'intorno dello zero...).*

ESERCIZIO 11. (Appello del 21/6/06). *Si consideri la funzione $f(x) = x^2 - \sin(\pi x)\, e^{-x}$.*

1. *Individuare un metodo di iterazione funzionale convergente linearmente alla radice positiva, $\alpha$, di $f(x)$.*
2. *Individuare un metodo di iterazione funzionale convergente quadraticamente alla radice $\beta = 0$, di $f(x)$.*

*In tutti i casi usare $tol = 1.e-6$ e calcolare l'errore assoluto.*

ESERCIZIO 12. (Laboratorio del 16/11/05)

1. *Si consideri la funzione $f(x) = x^2 - \log(x^2+2)$ di cui si vogliamo trovare gli zeri.*
   - *Individuare le due radici reali di $f(x) = 0$ e i corrispondenti intervalli separatori (che denoteremo con $I_{\alpha_1}$ e $I_{\alpha_2}$).*
   - *Si costruiscano due metodi convergenti di iterazione funzionale, le cui funzioni di iterazione sono $g_i(x)$, $i = 1, 2$. Determinare per ciascuno di essi il numero di iterazioni necessarie, l'ordine di convergenza e il fattore asintotico d'errore. Usare 50 come numero massimo di iterazioni e un opportuno test d'arresto con $tol = 1.e-5$*
2. *Data la funzione $f(x) = x^2 - 2x - \log(x)$, si studi la convergenza del metodo delle secanti applicato all'equazione $f(x) = 0$.*

   *Ricordo che la formula del metodo delle secanti è*

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})\frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}\ , \quad k \geq 1\ .$$

   *Si fornisca anche il plot della sequenza $\{x_i\}$ alle due radici reali di $f$. Si scelga $tol = 1.e-5$. Rifare quindi l'esercizio con il metodo di Newton (o delle tangenti).*

ESERCIZIO 13. `(Appello del 19/12/06)`. *Si consideri il metodo d'iterazione funzionale*

$$x_{i+1} = x_i + e^{1-x_i} - 1 \ .$$

*Provare, dapprima teoricamente e quindi numericamente usando* $tol = 1.e-9$, *che questo procedimento converge all'unico punto fisso della funzione d'iterazione. Calcolarne anche l'ordine di convergenza.*

ESERCIZIO 14. `(Appello del 26/9/05)`. *Si consideri la funzione* $f(x) = (x^2-1)^p \log(x),\ p \geq 1,\ x > 0$ *che ha in* $\alpha = 1$ *una radice multipla di molteplicità* $m = p+1$. *Nei casi* $p = 2, 4, 6$, *si determini* $\alpha$ *con i due seguenti metodi a meno di* $tol = 1.e-8$ *partendo da* $x_0 = 0.8$.

1. $$x_{k+1} = x_k - m_k \frac{f(x_k)}{f'(x_k)},\ k \geq 2 \quad con \quad m_k = \frac{x_{k-1} - x_{k-2}}{2x_{k-1} - x_k - x_{k-2}} \ . \tag{2.28}$$

2. $$x_{k+1} = x_k - m \frac{f(x_k)}{f'(x_k)} \ .$$

*Per ciascun metodo si determini il numero di iterazioni necessarie. Nel caso del primo metodo si faccia vedere che la formula per* $m_k$ *in (2.28) fornisce anche una stima della molteplicità di* $\alpha$.

ESERCIZIO 15. `(Appello del 23/3/05)`. *Si consideri l'equazione* $x = e^{-x}$.

- *Individuato un intervallo che contiene la radice, usando l'iterazione*

$$x_{n+1} = \frac{e^{-x_n} + x_n}{2}, \quad n \geq 0 \tag{2.29}$$

  *si determini la radice* $\alpha$ *dell'equazione data con* $tol = 1.e-6$.

- *Si prenda ora l'iterazione*

$$x_{n+1} = \frac{\omega e^{-x_n} + x_n}{1+\omega}, \quad n \geq 0,\ \ \omega \neq 0,\ \omega \neq -1 \ , \tag{2.30}$$

  *Determinata* $\alpha$, *graficamente si dica per quali valori di* $\omega$ *l'iterazione (2.30) converge più rapidamente di (2.29) (Sugg. si calcoli in* $\alpha$ *la derivata della funzione d'iterazione (2.30) )*

- *Qual è il valore ottimale di* $\omega$*?*

ESERCIZIO 16. *Si considerino le funzioni* $f_1(x) = \log(2/(3-x))$ *e* $f_2(x) = x^3 - 3$ .

- *Mediante il metodo di Newton determinare l'unica intersezione* $x^\star$ *di* $f_1(x) = f_2(x)$ *calcolando anche il numero di iterazioni.*

- *Visualizzare in scala semilogaritmica l'andamento dell'errore relativo usando la soglia* $tol = 1.e-9$.

ESERCIZIO 17. `(Appello del 21/12/05)`. *Si considerino le funzioni* $f_1(x) = \log(2|x|)$ *e* $f_2(x) = 1 - k\,x$, *k reale.*

1. *Aiutandosi con la grafica, dire quante soluzioni reali hanno le due funzioni per i seguenti valori* $k_1 = 2e^{-2} - 0.1$, $k_2 = k_1 + 0.3$ *e* $k_3 = 0$.
2. *Si consideri quindi* $k = 1$. *Studiare la convergenza dei seguenti metodi di iterazione funzionale all'unica radice* $\alpha$

   *(i)*
   $$x_{i+1} = 1 - \log(2|x_i|)\,,$$
   *(ii)*
   $$x_{i+1} = \frac{1}{2}\exp(1 - x_i)\,.$$

ESERCIZIO 18. *Si consideri la funzione* $f(x) = x^3 - 3\,e^x + 3$ *di cui si vogliamo trovare gli zeri.*

- *Individuare le due radici reali di* $f(x) = 0$ *e i corrispondenti intervalli separatori (che denoteremo con* $I_{\alpha_1}$ *e* $I_{\alpha_2}$*) e verificare che* $\alpha_1 < \alpha_2 = 0$.
- *Si determini* $\alpha_1$ *con il metodo delle secanti. Usare un opportuno test d'arresto con* $tol = 1.e-8$.
- `facoltativo:` *individuare un metodo di iterazione funzionale convergente ad* $\alpha_1$.

ESERCIZIO 19. `(Appello del 29/3/07)`. *Si consideri la funzione*

$$f(x) = 1.74\,\log(10\sqrt{x}) - \frac{4}{10} - \frac{1}{\sqrt{x}}\,.$$

- *Trovare l'intervallo* $[a, b]$ *che contiene l'unica radice* $\alpha$ *di* $f(x)$.
- *Costruire un metodo d'iterazione funzionale convergente in* $[a, b]$ *alla radice* $\alpha$. *Usare* $tol = 1.e-6$.

# Capitolo 3

# Soluzione di sistemi lineari

Prima di addentrarci nello studio dei metodi numerici, è doveroso introdurre le matrici e alcune strutture particolari di matrici nonchè alcuni concetti fondamentali quali la norma vettoriale e matriciale e il numero di condizionamento di una matrice.

## 3.1 Cose basilari sulle matrici

Le definizioni che qui forniamo sono relative a matrici a valori reali ma valgono similmente nel caso complesso con alcune piccole variazioni.

Una *matrice* (di numeri reali) è una tabella di $m \times n$ numeri disposti su $m$ righe e $n$ colonne. I numeri che compaiono nella tabella si chiamano *elementi* (della matrice). La loro individuazione avviene attraverso la loro posizione di riga e colonna. Ad esempio

$$A = \begin{bmatrix} 1 & 2 & 4 & 8 & 6 \\ 0 & -5 & 16 & -9 & 0.3 \\ 3 & 25 & 6 & 3 & 0.5 \end{bmatrix}$$

è una mtrice $3 \times 5$. L'elemento 16 essendo posizionato sulla seconda riga e terza colonna, verrà indicato con $a_{23}$.

In generale una matrice $A$ avente $m$ righe ed $n$ si indicherà con

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1j} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2j} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{ij} & \cdots & a_{in} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mj} & \cdots & a_{mn} \end{bmatrix}.$$

Il numero di righe o di colonne viene detto *ordine* o *dimensione della matrice*. Nel caso in cui $m = n$ si dice che la matrice è *quadrata di ordine n* altrimenti sarà detta *rettangolare*.

### 3.1.1 Operazioni aritmetiche con le matrici

- `Somma di matrici`. Siano $A$ e $B$ due matrici quadrate di ordine $n$ o in generale dello stesso tipo $m \times n$. Indicando con $C$ la matrice risultato dell'operazione, si ha:

$$C_{ij} = (A + B)_{ij} = A_{ij} + B_{ij} \ .$$

*Esempio.*

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ 2 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \\ 3 & 3 & 3 \end{bmatrix} .$$

- `Prodotto per uno scalare`. Sia $A$ una matrice quadrata di ordine $n$ o rettangolare $m \times n$. Preso un $\alpha \in \mathbb{R}$ si ha

$$(\alpha\, A)_{ij} = \alpha\, A_{ij} \ .$$

*Esempio.*

$$2 \begin{bmatrix} 1 & 3 & 4 \\ 0 & 1 & 5 \\ -1 & 7 & -8 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 6 & 8 \\ 0 & 2 & 10 \\ -2 & 14 & -16 \\ 2 & 0 & 0 \end{bmatrix} .$$

- `Prodotto di matrici`. La regola fondamentale è che il prodotto di matrici si fa `righe per colonne`. Pertanto perchè abbia senso richiederemo che il numero di colonne della prima matrice sia uguale al numero di righe della seconda matrice.

  Se $A$ e $B$ sono matrici quadrate di ordine $n$ il prodotto è sempre possibile. Invece se $A$ e $B$ sono rettangolari il numero delle colonne di $A$ deve coincidere con quello delle righe di $B$. As esempio, se $A$ è $n \times p$ e $B$ è $p \times m$ allora $C = A \times B$ avrà dimensione $n \times m$. Pertanto,

$$C_{ij} = \sum_{k=1}^{p} (A_{ik}\, B_{kj}) \ .$$

*Esempio.*

$$\begin{bmatrix} 1 & 3 & 2 \\ 0 & 0 & 1 \\ 1 & 2 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 4 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 14 & 6 \\ 0 & 1 \\ 9 & 5 \end{bmatrix} .$$

Elenchiamo le principali proprietà dell' operazione di somma e prodotto con matrici.

1. $A + 0 = 0 + A = A$ ovvero la matrice formata da tutti zeri è l'elemento neutro della somma.

2. $A + (-A) = 0$, ovvero esiste l'opposto di $A$ che è la matrice $-A$.
3. $(A + B) + C = A + (B + C)$, ovvero la somma è associativa.
4. $A + B = B + A$: la somma è commutativa. Purtroppo questa proprietà non vale per il prodotto: *il prodotto di matrici non è commutativo.*

$$\begin{bmatrix} 1 & 2 \\ 4 & -1 \end{bmatrix} \cdot \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 20 & -1 \end{bmatrix},$$

mentre

$$\begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 4 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 10 \\ 4 & -1 \end{bmatrix}.$$

5. $(AB)C = A(BC)$: associatività del prodotto di matrici.
6. $C(A + B) = CA + CB$: distributività del prodotto rispetto la somma.

Ci sono poi, alcune operazioni sulle matrici, tipiche dell'algebra delle matrici.

- `Somma diretta di matrici`. Siano $A$ e $B$ due matrici non necessariamente quadrate, la loro somma diretta, che si indica con $A \oplus B$ è

$$A \oplus B = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}.$$

*Esempio.*

$$\begin{bmatrix} 1 & 3 & 2 \\ 2 & 3 & 1 \end{bmatrix} \oplus \begin{bmatrix} 2 & 1 \\ 4 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 2 & 0 & 0 \\ 2 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In generale

$$\bigoplus_{i=1}^{k} A_i = A_1 \oplus A_2 \oplus \cdots \oplus A_k = \mathrm{diag}(A_1, \ldots, A_k).$$

In Matlab/Octave esiste la funzione `blkdiag` che permette di calcolare la somma diretta di matrici.

- `Prodotto diretto di matrici`. Siano $A$, $m \times n$ e $B$, $p \times q$ (in generale due matrici non necessariamente quadrate), il loro prodotto diretto, che si indica con $A \otimes B$ è

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ a_{21}B & \cdots & a_{2n}B \\ \vdots & & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

La matrice $C = A \otimes B$ ha quindi dimensione $mp \times nq$.

*Esempio.*

$$\begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 3 \\ 2 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 6 \\ 2 & 1 & 4 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 9 & 0 & 0 \\ 6 & 3 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix} .$$

- `Esponenziale di matrici quadrate`. Sia $A$, $n \times n$, l'esponenziale di $A$ si definisce come la serie di Taylor infinita

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!} . \tag{3.1}$$

Nel caso banale in cui $A$ sia $1 \times 1$ la serie coincide con l'usuale funzione esponenziale scalare. L' esponenziale di matrice viene usata soprattutto nel contesto della soluzione di sistemi di equazioni differenziali ordinarie. In Matlab/Octave la funzione `expm` consente di calcolare l'esponenziale di matrice mediante un'approssimazione di Padè, che consiste in un'approssimazione polinomiale razionale di matrici dell'espansione di Taylor (3.1) (cfr. [14, 17]).

Alcune strutture speciali sono le seguenti

- $A$ è detta `diagonale` se

$$\begin{pmatrix} a_{1,1} & & & 0 \\ 0 & a_{2,2} & & \\ \vdots & & \ddots & \\ 0 & & & a_{n,n} \end{pmatrix} , \quad a_{i,j} = 0, \ i \neq j .$$

- $A$ è detta `triangolare superiore` se

$$\begin{pmatrix} x & \cdots & \cdots & x \\ & x & \cdots & x \\ & 0 & \ddots & \vdots \\ & & & x \end{pmatrix} , \ a_{i,j} = 0, \ i > j .$$

- $A$ è detta `triangolare inferiore` se

$$\begin{pmatrix} x & & & \\ x & \ddots & 0 & \\ \vdots & \cdots & \ddots & \\ x & \cdots & \cdots & x \end{pmatrix} , \quad a_{i,j} = 0, \ i < j .$$

- $A$ è detta `tridiagonale` se

$$\begin{pmatrix} x & x & & & 0 \\ x & x & x & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & x \\ 0 & & & x & x \end{pmatrix}, \quad a_{i,j} = 0,\ |i-j| > 1\,.$$

  Si dirà che una matrice è `a banda` con banda di ampiezza $2s+1$ se gli elementi nulli sono quelli i cui indici soddisfano la disuguaglianza $|i-j| > s$.

- $A$ è detta avere la forma di matrice di `Hessenberg superiore` se

$$\begin{pmatrix} x & x & x & \cdots & x \\ x & x & x & \cdots & x \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & x & x \end{pmatrix}, \quad a_{i,j} = 0,\ i > j+1\,.$$

  Si dirà poi che $A$ ha la forma di `Hessenberg inferiore` se $a_{i,j} = 0,\ j > i+1\,.$

- $A$ si dice `a blocchi` se i suoi elementi sono a loro volta delle matrici. Ad esempio una matrice a blocchi $2 \times 2$ si indica come segue

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}.$$

- `Trasposta di una matrice`

  La matrice `trasposta` di $A$, denotata con $A^T$ è tale che $(A)_{ij}^T = A_{ji}$. La trasposizione gode delle seguenti proprietà.

$$(A^T)^T = A,\ (cA)^T = cA^T,\ (A+B)^T = B^T + A^T,\ (AB)^T = B^T A^T\,.$$

  Se $A$ e $B$ sono due matrici a blocchi, i cui blocchi hanno la stessa dimensione, la somma $A+B$ equivale alla matrice i cui elementi sono la somma dei rispettivi blocchi. Sempre nel caso di matrici a blocchi $2 \times 2$ avremo

$$A + B = \begin{pmatrix} A_{11}+B_{11} & A_{12}+B_{12} \\ A_{21}+B_{21} & A_{22}+B_{22} \end{pmatrix}.$$

  Anche l'operazione di trasposizione si applica a matrici a blocchi. Ad esempio

$$A^T = \begin{pmatrix} A_{11}^T & A_{21}^T \\ \\ A_{12}^T & A_{22}^T \end{pmatrix}.$$

Se $A^T = A$ allora $A$ è detta `simmetrica`. Quando $A^T = -A$, $A$ è detta `antisimmetrica`.

Osserviamo che la matrice trasposta esiste sia nel caso di matrici quadrate che rettangolari.

- `Inversa di una matrice`

  La matrice inversa di una matrice quadrata $A$ di ordine $n$, che si indica con $A^{-1}$, è tale che $A\,A^{-1} = A^{-1}\,A = I$. Valgono operazioni simili alla trasposta. Se $A$ e $B$ sono quadrate di ordine $n$ allora $(AB)^{-1} = B^{-1}A^{-1}$.

  Nel caso di matrici rettangolari, non si può definire l'inversa nel modo in cui siamo abituati, ma come vedremo più oltre nel contesto della soluzione di sistemi lineari sovra o sotto-determinati, si parlerà di *inversa generalizzata* o *pseudo inversa di Moore-Penrose* (vedi sezione 3.7).

**Definizione 7.** *Data $A$, diremo che $B$ è* `simile` *ad $A$ se esiste una matrice invertibile $P$ tale che*

$$P^{-1}AP = B\,.$$

### 3.1.2 Determinante e autovalori

Ad ogni matrice quadrata $A$ di ordine $n$, possiamo associare un numero detto *determinante* che denoteremo con $\det(A)$ oppure $|A|$. Se indichiamo con $\mathcal{M}$ l'algebra delle matrici quadrate di ordine $n$, allora il determinante *det* è una funzione da $\mathcal{M}$ a valori nei reali:

$$\begin{aligned} \det \;:\; & \mathcal{M} \to \mathbb{R} \\ & A \to \det(A) \end{aligned}$$

Per il calcolo del determinante ci si può avvalere della `regola di Laplace`

$$\det(A) = \begin{cases} a_{11} & n = 1 \\ \sum_{j=1}^{n} \tilde{A}_{i,j} a_{i,j}, \quad i = 1, ..., n & n > 1 \end{cases} \tag{3.2}$$

con $\tilde{A}_{i,j} = (-1)^{i+j}\det(A_{i,j})$ dove $A_{i,j}$ è la matrice ottenuta sopprimendo la $i$-esima riga e la $j$-esima colonna.

**Definizione 8.** *Un* `autovalore` *di una matrice $A$, è un numero $\lambda \in \mathbb{C}$ per cui esiste un vettore $\mathbf{x}$ non nullo per cui vale l'uguaglianza*

$$\lambda\mathbf{x} = A\mathbf{x}\,. \tag{3.3}$$

*Il vettore $\mathbf{x} \neq \mathbf{0}$ viene detto* `autovalore` *di $A$ associato all'autovalore $\lambda$.*

Il numero $\lambda$ è soluzione dell'equazione caratteristica

$$p_A(\lambda) = \det(A - \lambda I) = 0\,,$$

con $p_A(\lambda)$ che si chiama `polinomio caratteristico` della matrice $A$. Valgono inoltre le relazioni

$$\begin{aligned} \det(A) &= \prod_{i=1}^{n} \lambda_i\,, \\ \mathrm{tr}(A) &= \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} a_{i,i}\,, \end{aligned}$$

dove *tr* indica la `traccia` di $A$. Queste due uguaglianze si dimostrano facilmente osservando che il polinomio caratteristico $p_A(\lambda)$ è un polinomio monico di grado $n$ della forma

$$p_A(\lambda) = (-1)^n \lambda^n + (-1)^{n-1} \left( \sum_{i=1}^{n} a_{ii} \right) \lambda^{n-1} + \cdots + \det(A)\,.$$

Ricordando le relazioni tra le radici di un polinomio e i suoi coefficienti, si conclude.

In Matlab/Octave esistono le funzioni `det` e `trace`.

**Definizione 9.** *Una matrice simmetrica si dice* `definita positiva` *se per ogni vettore $x \neq 0$ la forma quadratica $x^T\,A\,x$ risulta essere maggiore di zero. Se $x^T\,A\,x \geq 0$ la matrice si dice semidefinita positiva.*

**Proposizione 3.** *Se $A$ è simmetrica definita positiva allora*

1. $|A_k| > 0,\ \forall\ k = 1, ..., n$, *cioè i minori principali di testa (incluso il determinante) sono positivi.*

2. $a_{i,i} > 0$.

3. $|a_{i,j}^2| < a_{i,i} a_{j,j},\ \ i \neq j$ , *ovvero l'elemento più grande sta sulla diagonale principale.*

4. *Gli autovalori di $A$ sono tutti positivi. Infatti se $\lambda$ è un autovalore, dalla definizione di una matrice simmetrica e definita positiva otteniamo la disuguaglianza*

$$0 < x^T\,A\,x = \lambda\,x^T\,x$$

*da cui si conclude essendo $x^T\,x = \sum_{i=1}^{n} x_i^2 > 0$ per ogni vettore $x$ non nullo.*

## 3.2 Norme di vettore e di matrice

Per ogni vettore $x \in \mathbb{R}^n$, possiamo definire la *norma* come una funzione

$$\|\cdot\| : \mathbb{R}^n \to \mathbb{R}_+ ,$$

avente le seguenti proprietà:

1. $\|x\| > 0, \ \forall\, x \neq 0, \ \|x\| = 0 \ \Leftrightarrow \ x = 0$
2. $\|c\,x\| = |c|\|x\|, \ \forall\, c \in \mathbb{R}$
3. $\|x + y\| \leq \|x\| + \|y\|, \ \forall\, x, y \in \mathbb{R}^n$ (disuguaglianza triangolare).

Una proprietà importante è che in uno spazio vettoriale di dimensione finita *tutte le norme vettoriali sono equivalenti*. Ovvero per ogni coppia di norme $\|\cdot\|^{(1)}$ e $\|\cdot\|^{(2)}$ esistono due costanti positive $m$ e $M$ t.c.

$$m\|x\|^{(2)} \leq \|x\|^{(1)} \leq M\|x\|^{(2)}, \ \forall\, x \in \mathbb{R}^n \, . \tag{3.4}$$

Gli esempi di norme vettoriali più usate sono:

(a) $\|x\|_\infty = \max\limits_{1\leq i\leq n} |x_i|$ (norma infinito)

(b) $\|x\|_1 = \sum\limits_{1\leq i\leq n} |x_i|$ (norma 1)

(c) $\|x\|_2 = \left( \sum\limits_{1\leq i\leq n} |x_i|^2 \right)^{1/2} = \sqrt{x^T x}$ (norma 2 o norma euclidea)

(d) $\|x\|_p = \left( \sum\limits_{1\leq i\leq n} |x_i|^p \right)^{1/p}$, $p \geq 1$ (norma p)

In Matlab/Octave, queste norme si determinano usando la funzione `norm(x,*)`, dove `*` potrà assumere i valori `1,2,inf,p`. Per default `norm(x)` è la norma 2.

Se $A \in \mathbb{R}^{n\times n}$, la sua norma è ancora una funzione $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}_+$, che soddisfa le seguenti proprietà

1. $\|A\| > 0, \ \forall\, A \neq 0 \, ; \|A\| = 0 \iff A = 0$

2. $\|c\,A\| = |c|\|A\|, \ \ \forall\, c \in \mathbb{R}$
3. $\|A + B\| \le \|A\| + \|B\|, \ \ \forall\, A, B \in \mathbb{R}^{n\times n}$ (disuguaglianza triangolare).
4. $\|A\,B\| \le \|A\|\,\|B\|, \ \ \forall\, A, B \in \mathbb{R}^{n\times n}$

L'ultima proprietà è caratteristica della norma di matrice. Anche per le norme di matrici vale un'equivalenza simile alla (3.4). Gli esempi di norme matriciali più usate sono:

(a) $\|A\|_\infty = \max\limits_{1\le i\le n} \sum\limits_{j=1}^{n} |a_{i,j}|$ (norma infinito o norma per righe)

(b) $\|A\|_1 = \max\limits_{1\le j\le n} \sum\limits_{i=1}^{n} |a_{i,j}|$ (norma 1 o norma per colonne)

(c) $\|A\|_F = \left( \sum\limits_{1\le i\le n} \sum\limits_{1\le j\le n} |a_{i,j}|^2 \right)^{1/2} = \sqrt{\mathrm{tr}(A\,A^T)}\,,$ (norma di Frobenius)

(d) $\|A\|_2 = \sqrt{\rho(A^T A)}$ (norma 2 o norma euclidea o norma spettrale)

Osserviamo che la norma euclidea si chiama anche *norma spettrale* poiché $\rho(A) = \max_{1\le i\le n} |\lambda_i|$, con $\lambda_i$ i-esimo autovalore di $A$, si chiama *raggio spettrale* della matrice $A$. Inoltre, se $A$ è simmetrica $\|A\|_2 = \rho(A)$ altrimenti $\|A\|_2 = \sigma_1(A)$, con $\sigma_1(A)$ il più grande *valore singolare* della matrice $A$ (per la definizione di valori singolari di una matrice rimandiamo al capitolo successivo). Infatti, nel caso in cui $A = A^T$ per il generico autovalore avremo: $\lambda(A\,A^T) = \lambda(A^2) = \lambda^2(A)$ e dunque $\rho(A) = \|A\|_2$. Pertanto nel caso di matrici simmetrice il raggio spettrale è una norma.

**Definizione 10.** *Data una norma di matrice e una vettoriale, diremo che esse sono* `compatibili o consistenti` *se*

$$\|Ax\| \le \|A\|\|x\|, \ \ \forall\, A \in \mathbb{R}^{n\times n},\ x \in \mathbb{R}^n\ .$$

Ad ogni norma di vettore possiamo associare una norma di matrice nel seguente modo

$$\|A\| := \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|} = \sup_{\|x\|=1} \|Ax\| \tag{3.5}$$

Questa viene detta *norma naturale* o *norma indotta*. Ne consegue che $\|Ax\| \le \|A\|\|x\|$, ovvero che una norma indotta è anche compatibile. Come esempio, è facile verificare ricorrendo alla definizione che per la matrice identica

$$\|I\| = \max_{\|x\|=1} \|Ix\| = 1$$

e che le norme $1, 2, \infty$ sono norme naturali indotte dalle corrispondenti norme vettoriali. L'unica norma matriciale che non è naturale è quella di Frobenius. Infatti $\|I\|_F = \sqrt{n}$.

Infine è interessante ricordare la seguente proprietà:

**Proposizione 4.** *Per ogni norma compatibile con la corrispondente norma vettoriale, si ha*

$$\rho(A) \leq \|A\| .$$

**Dim.** Sia $\lambda$ autovalore di $A$ associato all'autovettore $\mathbf{v} \neq 0$. Avremo

$$|\lambda| \|\mathbf{v}\| = \|\lambda \mathbf{v}\| = \|A\mathbf{v}\| \leq \|A\| \, \|\mathbf{v}\|$$

da cui $\rho(A) := \max_{1 \leq i \leq n} |\lambda_i| \leq \|A\|$. $\square$

## 3.3 Soluzione di sistemi lineari: generalità

Data $A \in \mathbb{R}^{n\times n}$ e il vettore $b \in \mathbb{R}^n$ il problema consiste nel determinare il vettore $x \in \mathbb{R}^n$ soluzione del sistema lineare

$$Ax = b\,. \tag{3.6}$$

Anzitutto la soluzione di (3.6) esiste *se e solo se* la matrice $A$ è *invertibile*, che significa che $\det(A) \neq 0$. Se $A$ è invertibile sappiamo che grazie alla *regola di Cramer* le componenti del vettore soluzione $x$ sono

$$x_i = \frac{\det(A_i)}{\det(A)}\,,$$

con $A_i$ che è la matrice ottenuta da $A$ sostituendo la colonna $i$-esima con il termine noto $b$.

In pratica con la regola di Cramer si calcolano $n+1$ determinanti. Considerato che il calcolo di un determinante (con la regola di Laplace) costa $\mathcal{O}(n^3)$ operazioni, allora determinare la soluzione del sistema con Cramer costa $\mathcal{O}(n^4)$ operazioni. Pensando di doverla applicare a sistemi di grandi dimensioni, ad esempio $n > 100$, il metodo diventa via via inapplicabile dal punto di vista del tempo di calcolo.

### 3.3.1 Condizionamento del problema

Analizziamo due situazioni: perturbazione del termine noto e perturbazione simultanea della matrice e del termine noto.

1. Sia $\delta b$ la quantità di cui perturbiamo il termine noto $b$. Questa perturbazione si ripercuoterà sulla soluzione cosicchè invece di $x$ otterremo la soluzione $x + \delta x$. Vogliamo vedere come $\delta x$ può essere legato a $\delta b$. Pertanto, da

   $$A(x + \delta x) = b + \delta b$$

   ricordando che $Ax = b$, otteniamo il sistema $A\delta x = \delta b$. Ora,

   $$\|\delta x\| = \|A^{-1}\delta b\| \leq \|A^{-1}\|\,\|\delta b\|\,,$$

   ma $\|b\| = \|Ax\| \leq \|A\|\,\|x\|$, pertanto

   $$\frac{\|\delta x\|}{\|A\|\|x\|} \leq \|A^{-1}\|\frac{\|\delta b\|}{\|b\|}\,.$$

   Per l'errore relativo abbiamo infine la maggiorazione

   $$\frac{\|\delta x\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|\delta b\|}{\|b\|}\,. \tag{3.7}$$

Definendo poi $\kappa(A) = \|A\|\|A^{-1}\|$ come il `numero di condizionamento` della matrice $A$, possiamo dedurre da (3.7) che il rapporto tra l'errore relativo sulla soluzione e quello sul termine noto è maggiorato dal numero di condizionamento della matrice. Più la matrice sarà malcondizionata e peggiore sarà la maggiorazione e quindi la perturbazione indotta sulla soluzione. $\kappa(A)$ è quindi un fattore di amplificazione dell'errore.

In Matlab la funzione `cond(A,p)` consente di calcolare il numero di condizionamento di $A$ in norma $p = 1, 2, \infty$. In Octave esiste il comando `cond(A)` che calcola il numero di condizionamento della matrice A in norma 2.

Facciamo notare che in alcuni testi, invece di $\kappa(A)$, si trovano le notazioni $\mu(A)$ o $\nu(A)$.

2. Se perturbiamo anche la matrice $A$ di una quantità $\delta A$, si può dimostrare che per l'errore relativo vale la maggiorazione

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \left( \frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right). \tag{3.8}$$

Nel caso in cui $\|\delta A\| \leq \dfrac{1}{2\|A^{-1}\|}$, in (3.8) si ha

$$\frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}} \leq 2\kappa(A).$$

Come dicevamo il numero di condizionamento di $A$ dà indicazioni sull'amplificazione dell'errore relativo sulla soluzione. Osservando che $\kappa(A) \geq 1$ (assume il valore 1 quando $A$ è la matrice identica), allora *più piccolo sarà $\kappa(A)$ e meglio condizionato risulterà il problema* della soluzione di un sistema lineare.

Diamo solo un paio di esempi che quantificano il concetto di matrice malcondizionata.

**1.** `Matrice di Hilbert, H`.

È una matrice simmetrica di ordine $n$ i cui elementi sono $H_{i,j} = 1/(i + j - 1)$, $1 \leq i, j \leq n$.

Si dimostra che $\kappa_2(H) \approx \mathrm{e}^{3.5n}$. Alcuni valori di $\kappa_2(H)$, sono riportati in Tabella 3.1.

| n | 2 | 6 | 10 |
|---|---|---|---|
| $\kappa_2(H)$ | 19.3 | $1.5\,10^7$ | $1.6\,10^{13}$ |

Tabella 3.1: Numero di condizionamento in norma 2 della matrice di Hilbert

In Matlab/Octave esiste la funzione `hilb(n)` che consente di definire la matrice di Hilbert di ordine $n$.

**2.** `Matrice di Vandermonde, V`.

È la matrice associata al problema d'interpolazione polinomiale su $n$ punti (distinti) $x_1, \cdots, x_n$. La matrice di Vandermonde di ordine $n$ è

$$V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & x_1 & \dots & x_1^{n-1} \\ \vdots & & & \vdots \\ 1 & x_n & \dots & x_n^{n-1} \end{pmatrix}.$$

Si dimostra che per *punti distinti* $x_i \neq x_j$, $\det(V) = \prod_{i \neq j}(x_i - x_j) \neq 0$ .

Circa il numero di condizionamento $\kappa_\infty(V)$ vale la pena ricordare il risultato di Gautschi e Inglese [11]. Sia $X = \{x_i\}$ di cardinalità $n$, l'insieme dei nodi d'interpolazione. Se $x_i > 0$ allora $\kappa_\infty(X) \geq (n-1)2^{(n-1)}$. Invece, se i nodi sono simmetrici rispetto l'origine si hanno i seguenti limiti inferiori:

$$\begin{aligned} \kappa_\infty(X) &\geq (n-2)2^{\frac{(n-2)}{2}} \quad \text{se } n \text{ pari} \\ \kappa_\infty(X) &\geq (n-3)2^{\frac{(n-3)}{2}} \quad \text{se } n \text{ dispari} . \end{aligned}$$

Anche per la matrice di Vandermonde esiste una funzione Matlab/Octave che si invoca come `V=vander(x)` dove `x` è un vettore e la matrice `V` è tale che $V_{i,j} = x_i^{n-j}$.

A completamento ricordiamo che Matlab contiene una *galleria* di matrici test nel cosidetto *Matrix Computational Toolbox (MCT)* di Nick Higham

`www.maths.manchester.ac.uk/~higham/mctoolbox/`.

Per ottenere la lista di tutte le matrici test disponibili basta usare il comando

```
[out1,out2,...]=gallery(matname,opt1,opt2,...)
```

## 3.4 Metodi diretti

Si tratta di metodi numerici che consentono di determinare la soluzione del sistema lineare, teoricamente in un numero finito di passi. Putroppo a causa degli inevitabili errori di rappresentazione e algoritmici, i metodi necessitano di alcune strategie implementative. I metodi diretti che studieremo in questo testo sono: il metodo di eliminazione di Gauss che dimostreremo essere equivalente alla fattorizzazione LU di $A$, il metodo di Cholesky che si applica quando $A$ è simmetrica e l'algoritmo di Thomas per matrici tridiagonali.

### 3.4.1 Il Metodo di Eliminazione di Gauss (MEG)

Dato il sistema $Ax = b$ il metodo di Gauss consiste di due passi principali:

(i) eliminazione;

(ii) sostituzione all'indietro.

L'obiettivo del passo di eliminazione è di trasformare la matrice $A$ in forma di matrice triangolare superiore allo scopo di ricavere la soluzione del sistema mediante appunto la sostituzione all'indietro (ovvero determinando dapprima $x_n$ e via via tutte le altre componenti del vettore $x$).

Dato il sistema

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots & +a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots & +a_{2n}x_n & = & b_2 \\ \vdots & \vdots & = & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots & +a_{nn}x_n & = & b_n \end{cases} \tag{3.9}$$

che indicheremo più compattamente con $A^{(1)}x = b^{(1)}$, dove l'apice ci ricorda il passo di eliminazione, avendo posto $A^{(1)} = A$. Ora se $a_{11}^{(1)} \neq 0$ la prima equazione può essere usata per ricavare $x_1$ e sostituirlo nelle rimanenti equazioni ottenendo un nuovo sistema $A^{(2)}x = b^{(2)}$ che avrà un solo elemento non nullo nella prima colonna

$$\begin{cases} a_{1,1}^{(2)}x_1 & +a_{1,2}^{(2)}x_2 + \cdots & +a_{1,n}^{(2)}x_n & = & b_1^{(2)} \\ 0 & +a_{2,2}^{(2)}x_2 + \cdots & +a_{2,n}^{(2)}x_n & = & b_2^{(2)} \\ \vdots & \vdots & \vdots & = & \vdots \\ 0 & +a_{n,2}^{(2)}x_2 + \cdots & +a_{n,n}^{(2)}x_n & = & b_n^{(2)} \end{cases} \tag{3.10}$$

In pratica per passare da $A^{(1)}$ ad $A^{(2)}$ si individua dapprima il *moltiplicatore*

$$m_{i,1} = \frac{a_{i,1}^{(1)}}{a_{1,1}^{(1)}}, \quad i = 2, \ldots, n$$

di modo che gli elementi di $A^{(2)}$ e $b^{(2)}$ saranno

$$a_{i,j}^{(2)} = \begin{cases} a_{i,j}^{(1)} & i = 1 \\ a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)} & i > 1 \end{cases}$$

$$b_{i}^{(2)} = \begin{cases} b_{i}^{(1)} & i = 1 \\ b_{i}^{(1)} - m_{i,1} b_{1}^{(1)} & i > 1 \end{cases}$$

Se $a_{2,2}^{(2)} \neq 0$ si può continuare con la seconda colonna e così via.

Pertanto per $1 \leq k \leq n-1$, se $a_{k,k}^{(k)} \neq 0$ avremo

$$m_{i,k} = \frac{a_{i,k}^{(k)}}{a_{k,k}^{(k)}}, \quad i = k+1, \ldots, n$$

e

$$a_{i,j}^{(k+1)} = \begin{cases} a_{i,j}^{(k)} & i \leq k \\ a_{i,j}^{(k)} - m_{i,k} a_{k,j}^{(k)} & i = k+1, \ldots, n \quad j = i, ..., n \end{cases}$$

$$b_{i}^{(k+1)} = \begin{cases} b_{i}^{(k)} & i \leq k \\ b_{i}^{(k)} - m_{i,k} b_{k}^{(k)} & i = k+1, \ldots n \end{cases}$$

Alla fine il sistema $A^{(n)}x = b^{(n)}$ avrà la matrice $A^{(n)}$ che sarà *triangolare superiore*

$$\begin{cases} a_{1,1}^{(n)} x_1 & +a_{1,2}^{(n)} x_2 + \cdots & +a_{1,n}^{(n)} x_n & = & b_1^{(n)} \\ & +a_{2,2}^{(n)} x_2 + \cdots & +a_{2,n}^{(n)} x_n & = & b_2^{(n)} \\ & \ddots & & & \vdots \\ & & a_{n,n}^{(n)} x_n & = & b_n^{(n)} \end{cases} \tag{3.11}$$

A questo punto si può applicare la *sostituzione all'indietro* e determinare il vettore soluzione. Infatti se

$$a_{i,i}^{(n)} \neq 0, \quad i = 1, \ldots, n\,, \tag{3.12}$$

allora

$$x_n = b_n^{(n)} / a_{n,n}^{(n)} \tag{3.13}$$

$$x_i = \frac{1}{a_{i,i}^{(n)}} \left\{ b_i^{(n)} - \sum_{j=i+1}^{n} a_{i,j}^{(n)} x_j \right\}, \quad i = n-1, \ldots, 1\,. \tag{3.14}$$

**Algoritmo di eliminazione e di sostituzione all'indietro**

I due passi del metodo di eliminazione di Gauss si possono discrivere da un punto di vista algoritmico, usando sempre la sintassi Matlab/Octave, come segue.

**Algoritmo 2. Eliminazione**

```
for i=1:n-1,
  for j=i+1:n,
     m=a(j,i)/a(i,i);
     for k=i:n,
       a(j,k)=a(j,k)-m*a(i,k);
     end
   b(j)=b(j)-m*b(i);
   end
end
```

Dal punto di vista della complessità, al passo $i$-esimo di eliminazione il costo, in termini di moltiplicazioni e divisioni, è

$$\underbrace{(n-i)}_{\text{ciclo su j}}\ \underbrace{(n-i+1)}_{\text{ciclo su k}}\ +\ \underbrace{(n-i)}_{\text{ciclo su j per i }b_j}\ =\ (n-i)(n-i+2)\,,$$

Pertanto, per i tre cicli `for`, la complessità totale sarà

$$\sum_{i=1}^{n-1}(n-i)(n-i+2)=\sum_{i=1}^{n-1}(n^2+2n-2(n+1)i+i^2)\,. \tag{3.15}$$

Ricordando le identità

$$\sum_{i=1}^{n} i=\frac{n(n+1)}{2},\quad \sum_{i=1}^{n} i^2=\frac{n(n+1)(2n+1)}{6} \tag{3.16}$$

sostituendo in (3.15) (con $n-1$ al posto di $n$) otteniamo

$$\sum_{i=1}^{n-1}(n^2+2n-2(n+1)i+i^2)=n^2-n+\frac{2n^3-3n^2+n}{6}=\frac{n^3}{3}+\frac{n^2}{2}-\frac{5n}{6}\ \approx\mathcal{O}(n^3)\,. \tag{3.17}$$

Dimenticando i termini di ordine inferiore a $n^3$, diremo che la complessità dell'algoritmo di eliminazione è $n^3/3$.

**Algoritmo 3. Sostituzione all'indietro**

```
for i=n:-1:1,
  sum=0;
     for j=i+1:n,
       sum=sum+ a(i,j)*x(j);
     end
   x(i)=(b(i)-sum)/a(i,i);
 end
```

Anche per la sostituzione all'indietro possiamo fare un calcolo della complessità. Come è facile vedere, per costruire `sum` si fanno $n-i$ moltiplicazioni. Pertanto la complessità totale è

$$\sum_{i=1}^{n}(n-i) = \frac{n(n-1)}{2}\,.$$

La complessità del metodo di Gauss si ottiene sommando la complessità dell'algoritmo di elminazione e quella dell'algoritmo di sostituzione

$$\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6} + \frac{n(n-1)}{2} = \frac{n^3}{3} + n^2 - \frac{4n}{3}\,.$$

In conclusione, l'algoritmo di Gauss richiede $\mathcal{O}(n^3/3)$ operazioni.

Esempio 13.

$$A := A^{(1)} = \begin{pmatrix} 11 & 4 & -6 \\ -7 & 17 & 9 \\ -1 & -4 & 6 \end{pmatrix}, \quad b = b^{(1)} = \begin{pmatrix} 9 \\ 19 \\ 1 \end{pmatrix},$$

- *Primo passo.* I moltiplicatori sono $m_{2,1} = -7/11,\ m_{3,1} = -1/11$.

$$A^{(2)} = \begin{pmatrix} 11 & 4 & -6 \\ 0 & \frac{215}{11} & \frac{57}{11} \\ 0 & -\frac{40}{11} & \frac{60}{11} \end{pmatrix}, \quad b^{(2)} = \begin{pmatrix} 9 \\ \frac{272}{11} \\ \frac{20}{11} \end{pmatrix},$$

- *Secondo passo.* Il moltiplicatore è $m_{3,2} = -8/43$.

$$A^{(3)} = \begin{pmatrix} 11 & 4 & -6 \\ 0 & \frac{215}{11} & \frac{57}{11} \\ 0 & 0 & \frac{276}{43} \end{pmatrix}, \quad b^{(3)} = \begin{pmatrix} 9 \\ \frac{272}{11} \\ \frac{276}{43} \end{pmatrix},$$

Con la sostituzione all'indietro troveremo che la soluzione è $x = (1,1,1)^T$.

**Strategia del pivot**

L'ipotesi su cui si basa il MEG è che al passo $k$ gli elementi diagonali siano in modulo diversi da zero, ovvero $|a_{k,k}^{(k)}| \neq 0$. Ma se accade che $a_{k,k}^{(k)} \approx 0$, si può applicare la stategia del *pivot parziale per righe* consistente nel ricercare nelle righe $k+1, \ldots, n$ (quelle sotto la diagonale) l'elemento in modulo più grande. Sia $r$ l'indice di riga corrispondente, quindi si scambierà la riga $r$ con la riga $k$ (sia nella matrice che nel termine noto).

In Matlab/Octave la ricerca dell'elemento più grande in modulo nella colonna $k$-esima, sotto la diagonale principale e il corrispondente scambio della riga $r$ con quella $k$, si realizza come segue:

```
[M,r]=max(abs(a(k+1:n,k)));
t=a(k,:); a(k,:)=a(r,:); a(r,:)=t
```

Con questa strategia si ha una riduzione dell' errore algoritmo e quindi maggiore stabilità. Infatti, detto

$$|a_{r,k}^{(k)}| = \max_{k \leq i \leq n} |a_{i,k}^{(k)}| \,,$$

gli elementi di $A^{(k+1)}$ saranno tali che

$$|a_{i,j}^{(k+1)}| = |a_{i,j}^{(k)} - m_{i,k} a_{k,j}^{(k)}| \leq |a_{i,j}^{(k)}| + |a_{k,j}^{(k)}| \,. \tag{3.18}$$

La disuguaglianza deriva dal fatto che per costruzione $|m_{i,k}| \leq 1$. Detto poi $a_M^{(k)} = \max_{1 \leq i,j \leq n} |a_{i,j}^{(k)}|$, da (3.18) otteniamo

$$a_M^{(n)} \leq 2a_M^{(n-1)} \leq 2^2 a_M^{(n-2)} \leq \cdots \leq 2^{n-1} a_M^{(1)} \,. \tag{3.19}$$

Pertanto la strategia del pivot parziale per righe garantisce maggiore stabilità al MEG. È da osservare che la maggiorazione (3.19) non è quasi mai raggiunta.

Vediamo come è è possibile implementare la tecnica del pivot parziale per righe, facendo uso di un vettore $p$ che memorizza solo gli scambi di righe. All'inizio $p_i = i, \;\; i = 1, ..., n$. Il significato di $p_i$ è il seguente: *$a_{i,j}$ è memorizzato nella posizione di indice di riga $p_i$ e colonna $j$* (e $b_i$ nella posizione indicata da $p_i$). Quando si scambia la riga $k$ con la riga $r$, si scambia $p_k$ e $p_r$ cosicchè l'indice di riga che contiene il pivot è $p_r$.

Esempio 14. Mettiamo in un'unica matrice, la matrice dei coefficienti, il vettore colonna del termine noto e il vettore degli scambi, come segue:

$$\left(\begin{array}{ccc|c|c} 2 & 3 & -1 & 5 & 1 \\ 4 & 4 & -3 & 3 & 2 \\ -2 & 3 & -1 & 1 & 3 \end{array}\right),$$

- *Primo passo.* L'elemento pivot che vale 4, si trova in riga 2. Pertanto scambieremo $p_2$ e $p_1$ e l'indice del pivot sarà 2. Otteniamo i moltiplicatori $m^{(1)} = 1/2$ e $m^{(2)} = -1/2$ e la nuova matrice
$$\left(\begin{array}{ccc|c|c} 0 & 1 & 1/2 & 7/2 & 2 \\ 4 & 4 & -3 & 3 & 1 \\ 0 & 5 & -5/2 & 5/2 & 3 \end{array}\right),$$
- *Secondo passo.* L'elemento pivot, che vale 5, si trova in riga 3. Pertanto dobbiamo scambiare $p_2$ e $p_3$. Il moltiplicatore è $m = 1/5$. Abbiamo allora
$$\left(\begin{array}{ccc|c|c} 0 & 0 & 1 & 3 & 2 \\ 4 & 4 & -3 & 3 & 3 \\ 0 & 5 & -5/2 & 5/2 & 1 \end{array}\right),$$

A questo punto, per applicare la sostituzione all'indietro, partiremo da $p_3 = 1$ ovvero dalla prima riga ricavando $x_3 = 3$. Poi si passa a $p_2 = 3$, quindi alla terza riga, ricavando $x_2$ dall'equazione $5x_2 - 15/2 = 5/2$ che ci dà $x_2 = 2$. Infine essendo $p_1 = 2$ dalla seconda equazione determineremo $x_1$ che, dopo aver risolto $4x_1 + 8 - 9 = 3$ mi darà $x_1 = 1$. È facile provare che il vettore $x = (1, 2, 3)^T$ è la soluzione del sistema lineare.

La tecnica del pivoting parziale si può applicare in alternativa alle colonne ottenendo il cosidetto *pivot parziale per colonne*.

Se invece la ricerca del massimo la facciamo su tutta la sottomatrice $A(k+1:n,\, k+1:n)$, ovvero quella di indici di riga e colonna compresi tra $k+1$ e $n$, allora parleremo di *pivoting totale*. In questo caso se $r$ e $s$ sono gl'indici di riga e colonna corrispondenti nella matrice $A^{(k)}$, allora dovremo scambiare la riga $k$ con la riga $r$ e la colonna $k$ con la colonna $s$.

### 3.4.2 Metodo di Gauss e la fattorizzazione LU

Faremo vedere che il MEG altro non è che la fattorizzazione della matrice del sistema $A = LU$ con $L$ triangolare inferiore con elementi diagonali tutti uguali a 1 e $U$ triangolare superiore. Ma prima di tutto enunciamo il teorema che ci garantisce quando è attuabile la fattorizzazione $LU$ di una matrice quadrata $A$.

**Teorema 2.** *Sia $A$ una matrice quadrata di ordine $n$ e siano $A_k$, $k = 1, \ldots, n$ le* `sottomatrici principali di testa`*. Ovvero*
$$A_1 = (a_{11}),\ A_2 = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$
$$A_k = \begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & & a_{kk} \end{pmatrix}$$

*cosicché $A_n = A$. Se $|A_k| \neq 0,\ \ k = 1, ..., n$ allora esiste unica la fattorizzazione di $A$ nella forma $LU$, con $L$ triangolare inferiore con elementi diagonali uguali a 1 e $U$ triangolare superiore. Altrimenti esiste una* `matrice di permutazione` *$P$ (i cui elementi sono 0 e 1) tale che $PA = LU$.*

Facciamo un paio di esempi che ci consentono di capire meglio il Teorema 2.

ESEMPIO 15.

La matrice

$$A = \begin{pmatrix} 1 & 2 & -1 \\ -1 & -1 & 2 \\ 1 & 1 & 2 \end{pmatrix} ,$$

soddisfa le ipotesi del Teorema 2. Si vede facilmente che $A$ può fattorizzarsi come

$$A = \begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 4 \end{pmatrix} .$$

ESEMPIO 16. La matrice

$$B = \begin{pmatrix} 1 & 2 & -1 \\ -1 & -2 & 0 \\ 1 & 1 & 2 \end{pmatrix}$$

non soddisfa le ipotesi del Teorema 2. Infatti

$$\det(B_2) = \det \begin{pmatrix} 1 & 2 \\ -1 & -2 \end{pmatrix} = 0 .$$

Però scambiando la seconda e terza riga mediante la matrice di permutazione

$$P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

allora avremo

$$PB = \begin{pmatrix} 1 & 2 & -1 \\ 1 & 1 & 2 \\ -1 & -2 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & -1 \\ 0 & -1 & 3 \\ 0 & 0 & -1 \end{pmatrix} .$$

È facile far vedere che si sarebbe ottenuta un'altra fattorizzazione se avessimo usato un'altra matrice di permutazione

$$P_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

ovvero scambiando la prima e la terza riga di $B$.

Ricordiamo che in Matlab/Octave esiste la funzione `lu` la cui chiamata completa si fa scrivendo il comando `[L,U,P] = lu(A)`, con ovvio significato delle matrici coinvolte. Se effettuassimo invece la chiamata `[L,U]=lu(A)`, la matrice `L` sarà triangolare inferiore ma `L=P*M` con `M` triangolare inferiore e `P` matrice di permutazione che serve al pivoting per righe di A.

### 3.4.3 Matrici elementari di Gauss

In questa breve sottosezione facciamo vedere chi sono realmente le matrici $L$ e $U$ della fattorizzazione $LU$. Il generico passo di eliminazione è infatti equivalente alla premoltiplicazione per la matrice $M_k = I - m_k\, e_k^T$ dove $I$ è la matrice identica e

$$m_k = \begin{pmatrix} 0 \\ \vdots \\ m_{k+1,k} \\ m_{k+2,k} \\ \vdots \\ m_{n,k} \end{pmatrix}, \quad e_k = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \leftarrow k$$

con $m_{i,k} = a_{i,k}^{(k)}/a_{k,k}^{(k)}$. Pertanto la $k$-esima matrice elementare di Gauss è

$$M_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & 0 & \\ & & 1 & & \\ 0 & & -m_{k+1,k} & & \\ & & \vdots & \ddots & \\ & & -m_{n,k} & & 1 \end{pmatrix}, \tag{3.20}$$

Quindi dopo gli $n-1$ passi di eliminazione

$$M_{n-1} \cdots M_2\, M_1\, A = A^{(n)}\,.$$

Ecco che le matrici $L$ e $U$ non sono altro che

$$L = (M_{n-1} \cdots M_1)^{-1} = M_1^{-1} \cdots M_{n-1}^{-1}\,; U = A^{(n)}\,. \tag{3.21}$$

È facile provare che

$$M_k^{-1} = \begin{pmatrix} 1 & & & & \\ & \ddots & & 0 & \\ & & 1 & & \\ 0 & & m_{k+1,k} & & \\ & & \vdots & \ddots & \\ & & m_{n,k} & & 1 \end{pmatrix}$$

da cui

$$L = \begin{pmatrix} 1 & & & 0 \\ m_{2,1} & 1 & & \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \\ m_{n,1} & \cdots & m_{n,n-1} & 1 \end{pmatrix} .$$

Infine nel caso generale in cui siano richieste ad ogni passo delle matrici di permutazione, ovvero

$$(M_{n-1}P_{n-1}) \cdots (M_1 P_1) A = U$$

posto $P = P_{n-1} \cdots P_1$ otterremo

$$L = P \left(M_{n-1}\, P_{n-1} \cdots M_1\, P_1\right)^{-1} .$$

**Osservazioni**

- Nota la fattorizzazione LU di una matrice $A$ o, più in generale, la fattorizzazione $LU = PA$, la soluzione del sistema lineare $Ax = b$ si farà risolvendo due sistemi triangolori. Ovvero,

  1. Risolvi il sistema $Lz = Pb$;
  2. Risolvi il sistema $Ux = z$.

  La soluzione di un sistema triangolare costa $\mathcal{O}(n^2)$. Complessivamente, come visto nella sezione precedente, la soluzione di un sistema lineare con il MEG o equivalentemente la fattorizzazione $LU$ della matrice $A$, costa $\mathcal{O}\left(\frac{n^3}{3}\right)$.

- Grazie alla fattorizzazione $LU$ di $A$ possiamo anche calcolare facilmente il determinante di $A$. Infatti, $|A| = |LU| = |L|\,|U| = \prod_{i=1}^{n} u_{i,i}$ essendo $|L| = 1$.

### 3.4.4 Il metodo di Cholesky

Si applica quando la matrice $A$ è *simmetrica definita positiva.* Vista la simmetria, $A$ si potrà fattorizzare nella forma

$$A = H\, H^T, \quad H = \begin{pmatrix} h_{1,1} & & & \\ \vdots & \ddots & 0 & \\ \vdots & & \ddots & \\ h_{n,1} & & \cdots & h_{n,n} \end{pmatrix}, \quad h_{i,i} > 0 .$$

Come determiniamo la matrice $H$? Basta fare il prodotto $H\,H^T$ e identificare gli elementi corrispondenti. Le formule per calcolare gli elementi di $H$ sono:

$$\begin{aligned} h_{1,1} &= \sqrt{a_{1,1}} \\ h_{i,j} &= \frac{1}{h_{j,j}}\left(a_{i,j} - \sum_{k=1}^{j-1} h_{i,k}h_{j,k}\right), \quad i = 2,\ldots,n\,;\; j = 1,\ldots,n \\ h_{i,i} &= \sqrt{a_{i,i} - \sum_{k=1}^{j-1} h_{i,k}^2}\,. \end{aligned}$$

Una possibile implementazione della fattorizzazione di Cholesky è nella funzione `cholesky.m` qui sotto riportata.

```
function [h]=cholesky(a)
%-----------------------------------
% input
% a=matrice iniziale (simm. def. +)
%
% output
% h, matrice tale che h*h'=a
%-----------------------------------
n=size(a);
h(1,1)=sqrt(a(1,1));
 for i=2:n,
  for j=1:i-1,
     s=0;
     for k=1:j-1,
        s=s+h(i,k)*h(j,k);
     end
      h(i,j)=1/h(j,j)*(a(i,j)-s);
   end
  h(i,i)=sqrt(a(i,i)-sum(h(i,1:i-1)^2));
end
return
```

È facile verificare che la complessità è metà del MEG ovvero $\mathcal{O}(n^3/6)$.

Un'altra interessante osservazione è che data la simmetria di $A$, la matrice $H$ può essere memorizzata nella stessa area di memoria di $A$ invece che allocare memoria aggiuntiva.

Infine, in Matlab/Octave la fattorizzazione di Cholesky si effettua usando il comando `H=chol(A)`.

### 3.4.5 Algoritmo di Thomas

Si consideri la matrice tridiagonale

$$A = \begin{pmatrix} a_1 & c_1 & & 0 \\ b_2 & a_2 & \ddots & \\ & \ddots & & c_{n-1} \\ 0 & & b_n & a_n \end{pmatrix}$$

Se la fattorizzazione $LU$ di A esiste, allora L e U sono due matrici bidiagonali (inferiore e superiore, rispettivamente) della forma

$$L = \begin{pmatrix} 1 & & & 0 \\ \beta_2 & 1 & & \\ & \ddots & \ddots & \\ 0 & & \beta_n & 1 \end{pmatrix}, \quad U = \begin{pmatrix} \alpha_1 & c_1 & & 0 \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ 0 & & & \alpha_n \end{pmatrix}.$$

I coefficienti incogniti si determinano imponendo l'uguaglianza $LU = A$, mediante il seguente *Algoritmo di Thomas*

$$\alpha_1 = a_1, \quad \beta_i = \frac{b_i}{\alpha_{i-1}}, \quad \alpha_i = a_i - \beta_i c_{i-1}, \quad i = 2, ..., n\,.$$

Data una matrice (sparsa) `A`, il comando Matlab/Octave `spdiags(A)`, che generalizza `diag`, viene usato in 5 modi differenti.

1. `B = spdiags(A)`: estrae tutte le diagonali non nulle di una matrice $A$, $m \times n$. B è una matrice $\min(m,n) \times p$ le cui colonne sono le $p$ diagonali non nulle di $A$.
2. `[B,d] = spdiags(A)`: restituisce un vettore `d`, `length(d)=p`, le cui componenti intere specificano le diagonali di $A$.
3. `B = spdiags(A,d)`: estrae le diagonali specificate da `d`.
4. `A = spdiags(B,d,A)`: rimpiazza le diagonali specificate da `d` con le colonne di `B`. L' output è in formato sparso.
5. `A = spdiags(B,d,m,n)`: crea una matrice sparsa $m \times n$ prendendo le colonne di B e ponendole lungo le diagonali specificate da `d`.

Esercizio 20. Come esercizio, si chiede di costruire una matrice tridiagonale `T` con i comandi Matlab/Octave

```
>> b=ones(10,1); a=2*b; c=3*b;
>> T=spdiags([b a c],-1:1,10,10);
```

Risolvere quindi il sistema `T*x=d` con l'algoritmo di Thomas, con `d` scelto cosicché si abbia `x=ones(10,1)`. Quante operazioni si risparmiano rispetto alla fattorizzazione classica $LU$ fatta con Gauss?

### 3.4.6 Raffinamento iterativo

Sia $\hat{x}$ la soluzione del sistema $Ax = b$ calcolata mediante l'algoritmo di Gauss, MEG. Il `raffinamento iterativo` detto anche *metodo post-iterativo* consiste dei 3 seguenti passi

1. calcola $r = b - A\hat{x}$;
2. risolvi il sistema $Ad = r$ usando per $A$ la fattorizzazione LU usata per risolvere $Ax = b$;
3. poni $y = \hat{x} + d$

ripeti finché

$$\frac{\|d\|}{\|y\|} > tol$$

dove $tol$ è una prefissata tolleranza. Ovvero ci si arresterà quando l'errore relativo rispetto alla soluzione $y$ risulta essere minore o uguale a $tol$.

**Nota:** di solito (cioè in assenza di errori di arrotondamento) bastano 1-2 iterazioni per convergere. Il metodo serve come stabilizzatore del MEG.

Una funzione Matlab/Octave che implementa l'algoritmo del `raffinamento iterativo` si può scrivere come segue.

```
function y=RafIter(x,L,U,tol)
%-----------------------------------------
% Inputs:
% x    = soluzione con MEG
% L, U = matrici della fattorizzazione LU di A
% tol  = tolleranza
%
% Output:
% y = soluzione ‘‘raffinata’’
%-----------------------------------------
kmax=20;   % numero massimo d'iterazioni
A=L*U;
b=A*x;     % determino il termine noto
r=b-A*x;   % residuo iniziale
% Risolvo il sistema Ad=r, sapendo che A=LU
z=L\r;
d=U\z;
```

```
y=x+d;
k=1;     %contatore delle iterazioni
while (norm(d)/norm(y)> tol & k<=kmax)
    x=y;
    r=b-A*x;
    z=L\r;
    d=U\z;
    y=x+d;
    k=k+1;
end
```

## 3.5 Calcolo dell'inversa di una matrice: cenni

Un metodo semplice e immediato di calcolo dell'inversa di una matrice $A$ non singolare è questo: risolvi gli $n$ sistemi non omogenei

$$A\,x_i = e_i, \quad i = 1, ..., n\ , \tag{3.22}$$

con $x_i$ vettore che rappresenta la $i$-esima colonna della matrice inversa e $e_i$ il vettore di tutti zeri eccetto che per la $i$-esima componente che vale 1. Purtroppo questa tecnica è molto costosa: $\mathcal{O}(n^4)$.

Ma il calcolo di $A^{-1}$ può essere fatto più semplicemente usando la fattorizzazione $A = LU$ e si ha $A^{-1} = U^{-1}L^{-1}$. Detta $Y = L^{-1}$, da $L$ possiamo ricavare $Y$ chiedendo che $LY = YL = I$ mediante le formule

$$\begin{cases} l_{j,j}y_{j,j} = 1 \\ \\ l_{j+1,j}y_{j,j} + l_{j+1,j+1}y_{j+1,j} = 0 \\ \\ \vdots \\ \\ l_{n,j}y_{j,j} + l_{n,j+1}y_{j+1,j} + \cdots + l_{n,n}y_{n,j} = 0 \end{cases}$$

valide per $j = 1, ..., n$. Ora ricordando che $L$ ha elementi diagonali unitari, dalle relazioni precedenti possiamo ricavare i valori di $Y$ come segue

```
for j=1:n,
  y(j,j)=1;
end
for j=1:n-1,
  for i=j+1:n,
```

```
    s=0;
    for k=j:i-1,
      s=s+l(i,k)*y(k,j);
    end
   y(i,j)=-s;
  end
end
```

In maniera analoga si procede per il calcolo di $Z = U^{-1}$.

Un metodo piú efficiente è il seguente (cfr. Du Croz J., Higham N. *IMA J. Numer. Anal. 12:1-19*, 1992). Esso risolve l'equazione $UXL = I$, supponendo $X$ parzialmente nota ad ogni step. L'idea é di partizionare le matrici $X$,$L$ e $U$ come segue:

$$U = \begin{pmatrix} u_{1,1} & \mathbf{u}_{1,2}^T \\ \mathbf{0} & U_{2,2}^T \end{pmatrix} \quad X = \begin{pmatrix} x_{1,1} & \mathbf{x}_{1,2}^T \\ \mathbf{x}_{2,1} & X_{2,2}^T \end{pmatrix} \quad L = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{l}_{2,1} & L_{2,2}^T \end{pmatrix}$$

dove i blocchi (1,1) sono scalari e la sottomatrice $X_{2,2}$ si assume già nota. Quindi il resto di $X$ si calcola risolvendo le equazioni:

$$\begin{aligned} x_{2,1} &= -X_{2,2}^T \mathbf{l}_{2,1} \\ x_{1,2}^T &= -\mathbf{u}_{1,2}^T X_{2,2}^T / u_{1,1} \\ x_{1,1} &= \tfrac{1}{u_{1,1}} - \mathbf{x}_{1,2}^T \mathbf{l}_{2,1} \end{aligned} .$$

In maniera algoritmica, usando notazioni Matlab/Octave, si può sintetizzare come segue, facendo attenzione al fatto che il blocco $X(k+1:n, k+1:n)$ si assume già noto. All'inizio dovremo conoscere l'elemento $x_{n,n}$ che si può determinare dall'equazione $UXL = I$.

```
for k=n-1:-1:1,
   X(k+1:n,k)=-X(k+1:n,k+1:n)*L(k+1:n,k)
   X(k,k+1:n)=-U(k,k+1:n)*X(k+1:n,k+1:n)/U(k,k)
   X(k,k)=1/U(k,k)-X(k,k+1:n)*L(k+1:n,k)
end;
```

ESERCIZIO 21. Scrivere degli scripts Matlab che implementano i metodi su descritti. Come matrice $A$ si consideri la matrice ottenuta usando la funzione `gfpp` del Toolbox *The Matrix Computational Toolbox (MCT)* di N. Higham (www.maths.manchester.ac.uk/∼higham/mctoolbox/). La funzione può essere usata con la sintassi `A=gfpp(n)` generando una matrice il cui fattore di crescita degli elementi è $2^{n-1}$, come per l'eliminazione gaussiana con pivoting parziale per righe e/o colonne.

- Verificare che la matrice `gfpp(n)` dà un fattore di crescita per MEG con pivoting parziale pari a $2^{n-1}$.

- Sempre nel MCT Toolbox, esiste la funzione `gep` che data una matrice, anche rettangolare, applica l'eliminazione gaussiana con pivoting *parziale* o *completo*, restituisce , tra le altre informazioni, la fattorizzazione LU e il fattore di crescita dei suoi elementi. Far vedere che se si usa la matrice `gep(gfpp(n),'c')` il fattore di crescita risulta uguale a 2.

## 3.6 Metodi iterativi

La filosofia di questi metodi sta nell'approssimare la soluzione del sistema lineare $Ax = b$ con una *successione di vettori* $\{x_k,\ k \geq 0\}$, a partire da un vettore iniziale $x_0 \in \mathbb{R}^n$, con l'obiettivo che converga verso la soluzione $x$ del sistema. A differenza dei metodi diretti, in questi metodi non si altera la struttura della matrice e pertanto sono utilizzati prevalentemente quando la matrice è sparsa.

La matrice $A$ di ordine $n$, che supponiamo sia non singolare, si può decomporre come

$$A = M - N$$

con la richiesta che $\det(M) \neq 0$ e facilmente invertibile. Pertanto

$$Mx - Nx = b \tag{3.23}$$
$$Mx = Nx + b \tag{3.24}$$
$$x = M^{-1}Nx + M^{-1}b \tag{3.25}$$

da cui, ponendo $P = M^{-1}N$ e $q = M^{-1}b$, la soluzione di $Ax = b$ è ricondotta al sistema $x = Px + q$. Scelto $x^{(0)}$, costruiremo la successione

$$x^{(i+1)} = Px^{(i)} + q,\ \ i = 0, 1, \ldots \tag{3.26}$$

**Definizione 11.** *La successione $\{x^{(i)}\}$ si dirà convergente al vettore $x$, e si scrive*

$$\lim_{i\to\infty} x^{(i)} = x\,,$$

*se per $i \to \infty$ le componenti di $x^{(i)}$ convergono verso le corrispondenti componenti di $x$.*

La (3.26) rappresenta un metodo iterativo per la soluzione di $Ax = b$, con $P$ che si chiama *matrice d'iterazione.*

**Definizione 12.** *Un metodo iterativo si dice* `convergente`*, se per ogni vettore iniziale $x^{(0)}$ la successione $\{x^{(i)}\}$ è convergente.*

Esempio 17. Siano

$$P = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 2 \end{pmatrix},\ \ q = 0,\ \text{con } x = 0\,.$$

Partendo da $x^{(0)} = (1, 0, 0)^T$ costruiremo la successione

$$x^{(1)} = P \cdot x^{(0)} = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ 0 \\ 0 \end{pmatrix}$$

$$x^{(2)} = P \cdot x^{(1)} = P^2 \cdot x^{(0)} \begin{pmatrix} \frac{1}{2^2} & 0 & 0 \\ 0 & \frac{1}{2^2} & 0 \\ 0 & 0 & 2^2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{2^2} \\ 0 \\ 0 \end{pmatrix}$$

e continuando otterremo

$$x^{(i)} = \begin{pmatrix} \frac{1}{2^i} \\ 0 \\ 0 \end{pmatrix} .$$

Pertanto per $i \to \infty$ la successione converge verso $x$. Si verifica facilmente che partendo da $x^{(0)} = (0,1,1)^T$ si avrebbe $x^{(i)} = \left(0, \frac{1}{2^i}, 2^i\right)^T$ e quindi una successione divergente.

A questo punto dobbiamo chiarire sotto quali condizioni il metodo iterativo risulta essere convergente.

**Teorema 3.** *Condizione* `necessaria` *per la convergenza è che esista una norma matrice indotta* $\|\cdot\|$ *per la quale risulta* $\|P\| < 1$.

**Dim.** Sia $e^k = x^{(k)} - x$ l'errore al passo $k$. Abbiamo

$$e^k = x^{(k)} - x = Px^{(k-1)} - q - Px + q = P(x^{(k-1)} - x) = Pe^{k-1}, \;\; k = 1, 2, ....$$

Ma $Pe^{k-1} = \cdots = P^{k-1}e^{(0)}$. Da cui

$$\|e^k\| \le \|P^k\| \|e^0\| \le \|P\|^k \, \|e^0\| .$$

Se quindi $\|P\| < 1$, allora $\lim_{k\to\infty} \|P\|^k = 0$ e anche $\|e^k\| \to 0 \;\; \forall\, k$. Per la continuità della norma concludiamo che $\lim_{k\to\infty} e^k = 0$ da cui l'asserto. □

Ricordando che vale

$$\rho(P) \le \|P\|,$$

per ogni norma indotta, la *condizione necessaria e sufficiente* per la convergenza di un metodo iterativo è contenuta nel seguente teorema.

**Teorema 4.** *Sia $P$ di ordine $n$. Allora*

$$\lim_{k\to\infty} P^k = 0 \iff \rho(P) < 1 .$$

◇◇

Prima di passare ai metodi, concludiamo questa parte sulle generalità dicendo quando numericamente consideriamo convergente un metodo iterativo. Fissata una tolleranza $\epsilon$ e indicato un numero massimo d'iterazioni `kmax`, il test d'arresto che valuteremo sarà

$$\|x^{(k)} - x^{(k-1)}\| \le \epsilon \|x^{(k)}\| \;\vee\; k > kmax$$

ovvero $x_k$ sarà una buona approssimazione di $x$ quando l'errore relativo è sotto una prefissata tolleranza. Ma il metodo si arresta anche quando $k > kmax$. In quest'ultimo caso molto probabilmente avremo fallito e onde evitare che si iteri all'infinito è buona norma inserire questo ulteriore controllo.

Ma possiamo anche fare le seguenti considerazioni. I metodi iterativi, per la soluzione di un sistema lineare $Ax = b$, teoricamente richiedono un numero infinito di iterazioni. Nella pratica ciò non è ragionevole poiché invece che $x$ ci si accontenta di una sua approssimazione $\tilde{x}$ o più concretamente di $x^k$, l'iterata ad un certo passo $k$ del metodo, per la quale l'*errore* sia inferiore ad una prescelta tolleranza $\epsilon$. Ma l'errore è a sua volta una quantità incognita perchè dipende dalla soluzione *esatta*. Nella pratica ci si rifà a degli stimatori dell'errore *a posteriori*.

(a) Un primo stimatore è il *residuo* ad ogni iterazione

$$r^k = b - Ax^{(k)}\,.$$

In tal caso ci arresteremo in corrispondenza a quel $k_{min}$ tale che

$$\|r^{k_{min}}\| \le \epsilon\|b\|\,. \tag{3.27}$$

Infatti, ricordando che $Ax = b$, la (3.27) altro non è che l'errore relativo

$$\left\|\frac{x - x^k}{x}\right\| = \left\|\frac{Ax - Ax^k}{Ax}\right\|\,.$$

Quindi, l'errore relativo

$$\frac{\|x - x^{k_{min}}\|}{\|x\|} = \frac{\|A^{-1}(b - A\,x^{k_{min}})\|}{\|x\|} \le \frac{\|A^{-1}\|\|r^{k_{min}}\|}{\|x\|} \le \epsilon\kappa(A)\,,$$

dove l'ultimo passaggio si ottiene moltiplicando numeratore e denominatore per $\|b\|$ e ricordando che

$$\frac{\|b\|}{\|x\|} \le \|A\|\,.$$

Perció, il controllo sul residuo ha senso solo se $\kappa(A)$, il numero di condizionamento della matrice $A$, è ragionevolmente piccolo.

(b) Alternativamente si può calcolare il cosidetto *incremento* $\delta^k = x^{(k+1)} - x^{(k)}$. In tal caso il metodo si arresterà al passo $k_{min}$ per cui

$$\|\delta^{k_{min}}\| \le \epsilon\|b\|\,.$$

Nel caso in cui la matrice di iterazione $P$ (non la matrice del sistema!) è simmetrica e definita positiva, posto $e^k = x^k - x$, in norma euclidea si avrà

$$\|e^k\| = \|e^{k+1} - \delta^k\| \le \|P\|\|e^k\| + \|\delta^k\| = \rho(P)\|e^k\| + \|\delta^k\|\,.$$

Per la convergenza, $\rho(P) < 1$, avremo alla fine

$$\|e^k\| \leq \frac{1}{1-\rho(P)}\|\delta^k\|\,. \tag{3.28}$$

Nota: se $P$ non è simmetrica e definita positiva si arriva alla stessa conclusione con $\|P\|$ al posto di $\rho(P)$.

In conclusione: il controllo sull'incremento è un buon stimatore quanto più $\rho(P) \ll 1$.

### 3.6.1 I metodi di Jacobi e Gauss-Seidel

Anzitutto facciamo alcune posizioni. Data la matrice quadrata $A$ indichiamo con $D$ la matrice dei valori diagonali di $A$, ovvero $d_i = a_{i,i}$ e con $B$ e $C$ le matrici triangolari inferiori e superiori rispettivamente ottenute nel seguente modo

$$b_{i,j} = \begin{cases} -a_{i,j} & i > j \\ 0 & i \leq j \end{cases} \qquad c_{i,j} = \begin{cases} 0 & i \geq j \\ -a_{i,j} & i < j \end{cases} ,$$

con $A = D - (B + C)$.

Nel `metodo di Jacobi` le matrici $M$ e $N$ prima definite sono

$$M = D,\ \ N = B + C\,.$$

Pertanto se $a_{i,i} \neq 0$, $\forall i$, allora $M$ è non singolare. La matrice di iterazione di Jacobi è

$$J = M^{-1}N = D^{-1}(B + C)\,.$$

Il metodo iterativo di Jacobi si può allora scrivere in termini vettoriali come

$$x^{(k)} = Jx^{(k-1)} + D^{-1}b,\ \ k \geq 1\,, \tag{3.29}$$

o per componenti come

$$x_i^{(k)} = \frac{1}{a_{i,i}}\left\{ - \sum_{j=1, j\neq i}^{n} a_{i,j}x_j^{(k-1)} + b_i \right\},\ i = 1, \ldots, n. \tag{3.30}$$

Nota

$$J = \begin{pmatrix} 0 & -\frac{a_{1,2}}{a_{1,1}} & \cdots & -\frac{a_{1,n}}{a_{1,1}} \\ -\frac{a_{2,1}}{a_{2,2}} & 0 & \cdots & -\frac{a_{2,n}}{a_{2,2}} \\ & & \ddots & \\ -\frac{a_{n,1}}{a_{n,n}} & & & 0 \end{pmatrix}.$$

Nel `metodo di Gauss-Seidel`, o semplicemente `G-S`, le matrici $M$ e $N$ prima definite sono

$$M = D - B, \;\; N = C\,.$$

La matrice di iterazione di Gauss-Seidel è

$$G = M^{-1}N = (D-B)^{-1}C\,.$$

Osserviamo che

$$\begin{aligned} x^{(k)} &= (D-B)^{-1}Cx^{(k-1)} + (D-B)^{-1}b \\ (D-B)x^{(k)} &= Cx^{(k-1)} + b \\ Dx^{(k)} &= Bx^{(k)} + Cx^{(k-1)} + b \end{aligned}$$

da cui otteniamo che, in termini vettoriali, il metodo di G-S si può scrivere come

$$x^{(k)} = D^{-1}Bx^{(k)} + D^{-1}Cx^{(k-1)} + D^{-1}b, \;\; k \geq 1\,, \tag{3.31}$$

o per componenti come

$$x_i^{(k)} = \frac{1}{a_{i,i}} \left\{ -\sum_{j=1}^{i-1} a_{i,j}x_j^{(k)} - \sum_{j=i+1}^{n} a_{i,j}x_j^{(k-1)} + b_i \right\}, \; i = 1, \ldots, n. \tag{3.32}$$

Dalle equazioni (3.29) e (3.31) si comprende perchè il metodo di Jacobi viene anche detto degli *spostamenti simultanei* mentre quello di G-S degli *spostamenti successivi*. Ma il vantaggio di G-S rispetto a Jacobi sta soprattutto nel poter memorizzare le componenti di $x^{(k)}$ nella stessa area di memoria di $x^{(k-1)}$.

Prima di discutere delle condizioni sotto le quali i metodi di Jacobi e G-S convergono, premettiamo alcune definizioni.

**Definizione 13.** *Una matrice $A$ si dice* `diagonalmente dominante per righe` *(o anche a predominanza diagonale per righe) se*

$$|a_{i,i}| \geq \sum_{j=1, j\neq i}^{n} |a_{i,j}| \tag{3.33}$$

*ed esiste un indice $s$ per cui la disuguaglianza vale in senso stretto. La matrice si dice invece* `diagonalmente dominante in senso stretto` *per righe (o a predominanza diagonale stretta per righe) se la (3.33) vale per ogni $i = 1, ..., n$.*

Analoga definizione vale per colonne.

**Definizione 14.** *Un grafo orientato si dice* `fortemente connesso` *se per ogni* $1 \leq i, j \leq n$, $i \neq j$ *esiste un cammino orientato che parte da* $p_i$ *ed arriva a* $p_j$ *(con* $p_i, p_j$ *nodi del grafo).*

Data una matrice $A$, il grafo ad essa associato si ottiene definendo tanti nodi $p_i$ quanti il numero $n$ (dimensione della matrice) con archi corrispondenti agli elementi non nulli di $A$. Ovvero, se $a_{i,j} \neq 0$ allora si disegnerà un arco che va da $p_i$ a $p_j$.

**Definizione 15.** *Una matrice* $A$ *si dice* `riducibile` *se e solo se il suo grafo orientatato non è fortemente connesso, altrimenti si dice* `irriducibile`.

Esempio 18. La matrice

$$A = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 2 & 3 & -2 & 1 \\ -1 & 0 & -2 & 0 \\ 1 & -1 & 1 & 4 \end{pmatrix}$$

ha il grafo che non è fortemente connesso. Infatti non esiste un arco orientato che va da $p_1$ a $p_4$.

Vale il seguente Teorema.

**Teorema 5.** *Sia* $A = M - N$ *la decomposizione di* $A$ *che, nel caso di Jacobi equivale ad* $M = D$, $N = B + C$, *mentre nel caso di G-S,* $M = D - B$ *e* $N = C$. *Se una delle seguenti ipotesi è verificata*

*(a)* $A$ *è strettamente diagonalmente dominante per righe o per colonne;*

*(b)* $A$ *è diagonalmente dominante e irriducibile;*

*allora* $\rho(M^{-1}N) < 1$ *e quindi i metodi di Jacobi o di G-S sono convergenti.*

Facciamo vedere che se vale (a) allora il metodo di Jacobi converge. Dobbiamo provare che $\rho(P_J) < 1$ con $P_J = D^{-1}(B + C)$ matrice d'iterazione di Jacobi. Dapprima osserviamo che, essendo $A$ diagonalmente dominante in senso stretto, non ha elementi diagonali nulli. Siano $\lambda$ e $x$ un generico autovalore e il corrispondente autovettore di $P_J$, ovvero, per componenti, $\sum_{j=1}^{n} p_{i,j}x_j = \lambda\, x_i, \quad i = 1, \ldots, n$. Possiamo assumere che $\max_{1\leq i\leq n} |x_i| = 1$. Sia $k$ l'indice in cui viene assunto il massimo. Avremo

$$|\lambda| = \left| \sum_{j\neq k, j=1}^{n} p_{k,j}x_j \right| \leq \sum_{j=1, j\neq k}^{n} \left| \frac{a_{i,j}}{a_{k,k}} \right| < 1$$

e vista la generalità di $\lambda$ si conclude che $\rho(P_J) < 1$. □

Vale anche il seguente risultato.

**Teorema 6.** *Sia $A$ tridiagonale di dimensione $n$ con $a_{i,i} \neq 0, \quad i = 1, \ldots, n$. Allora i metodi di Jacobi e di Gauss-Seidel sono o entrambi convergenti o entrambi divergenti. Se convergono, Gauss-Seidel converge più velocemente di Jacobi e si ha*

$$\rho(P_{GS}) = \rho^2(P_J) \ .$$

Esempio 19. La matrice

$$A = \begin{pmatrix} 4 & -1 & 1 & 1 \\ 0 & -4 & -1 & 1 \\ -1 & -1 & 4 & 1 \\ 1 & -1 & 0 & 4 \end{pmatrix}$$

ha il grafo che è fortemente connesso essendo diagonalmente dominante in senso stretto per colonne. Le matrici d'iterazione di Jacobi (J) e G-S (G) sono

$$J = \frac{1}{4}\begin{pmatrix} 0 & -1 & 1 & 1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & 0 & 1 \\ 1 & -1 & 0 & 0 \end{pmatrix}, \quad G = \frac{1}{16}\begin{pmatrix} 0 & 4 & -4 & -4 \\ 0 & 0 & -4 & 4 \\ 0 & 1 & -2 & -4 \\ 0 & -1 & 0 & 2 \end{pmatrix} .$$

È facile vedere che $\rho(J) \approx 0.4 < 1$ come pure $\rho(G) \approx 0.18 < 1$. Si noti inoltre che $\rho(G) < \rho(J)$. Pertanto sia il metodo di Jacobi che di G-S convergono.

Nel prossimo esempio facciamo vedere che se $A$ non è strettamente diagonalemente dominante, ma solo diagonalmente dominante, non è detto che il metodo di Jacobi e di G-S siano convergenti.

Esempio 20. La matrice

$$A = \begin{pmatrix} -4 & -1 & 1 & 1 \\ 0 & -4 & 0 & -4 \\ 1 & 1 & 4 & 1 \\ 0 & -4 & 0 & 4 \end{pmatrix}$$

È facile vedere che $\rho(J) = \rho(G) = 1$, pertanto sia il metodo di Jacobi che di G-S *non* convergono.

Il codice Matlab/Octave, `jacobi.m` in Appendice C è una implementazione del metodo di Jacobi, mentre `GuassSeidel.m` è il codice per il metodo di G-S. Il codice `GaussRil.m`, è più generale e può essere utilizzato anche per il metodo SOR che viene presentato nel prossimo paragrafo. Infatti, come faremo vedere, il metodo di G-S è un particolare metodo di rilassamento.

### 3.6.2 Il metodo SOR o di rilassamento

La filosofia di questo metodo sta nel determinare un parametro $\omega$ di accelerazione della convergenza del metodo di G-S. Partiamo considerando l'uguaglianza

$$\omega Ax = \omega b, \;\; \omega \neq 0, \; \omega \in \mathbb{R}\,.$$

Usando il fatto che vale $\omega A = M - N$, ricordando lo splitting $A = D - B - C$ e osservando che $\omega(D - B - C) + D - D = M - N$, una scelta per le matrici $M$ e $N$ è la seguente

$$M = D - \omega\, B\,, \;\; N = (1-\omega)D + \omega\, C\,.$$

Come si vede immediatamente, quando $\omega = 1$ , il predetto splitting equivale al metodo di G-S.

Se $\det(M) \neq 0$ allora ricaviamo il metodo

$$x^{(k)} = (D - \omega B)^{-1}\left[(1-\omega)D + \omega C\right] x^{(k-1)} + \omega(D - \omega B)^{-1} b\,, \;\; k = 0, 1, \ldots \tag{3.34}$$

con matrice d'iterazione, dipendente da $\omega$, data da

$$H(\omega) = (D - \omega B)^{-1}\left[(1-\omega)D + \omega C\right]\,. \tag{3.35}$$

Dalla (3.34) ricaviamo

$$\begin{aligned} Dx^{(k)} - \omega B x^{(k)} &= (1-\omega)Dx^{(k-1)} + \omega C x^{(k-1)} + \omega b\,, \\ x^{(k)} &= (1-\omega)x^{(k-1)} + \omega D^{(-1)}\left[Bx^{(k)} + Cx^{(k-1)} + b\right]\,, \end{aligned}$$

che per componenti diventa

$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \omega\left[b_i - \sum_{j=1}^{i-1}\frac{a_{i,j}}{a_{i,i}}x^{(k)} - \sum_{j=i+1}^{n}\frac{a_{i,j}}{a_{i,i}}x^{(k-1)}\right]\,,\; i = 1, ..., n\,. \tag{3.36}$$

Facciamo notare come il termine dentro parentesi quadre rappresenti la soluzione al passo $k$ ottenuta con il metodo di G-S. Pertanto la componente $i$-esima calcolata con il metodo SOR si può scrivere

$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \omega x_{i,GS}^{(k)}$$

che, come prima osservato, coincide con il metodo di G-S per $\omega = 1$. Il parametro $\omega$ serve ad accelerare la convergenza del metodo del metodo iterativo di G-S.

**Teorema 7.** *Condizione necessaria per la convergenza del metodo SOR è che*

$$0 < \omega < 2\,. \tag{3.37}$$

1. *Se $A$ è simmetrica definita positiva e $\omega$ soddisfa la (3.37) allora SOR converge, ovvero la condizione è anche sufficiente.*

2. *Se $A$ è tridiagonale, vale la (3.37) e gli autovalori della matrice d'iterazione di Jacobi, $J$, sono reali e t.c. $\rho(J) < 1$, allora esiste uno e uno solo $\omega_0$ t.c.*

$$\rho(H(\omega_0)) = \min_{0<\omega<2} \rho(H(\omega)) \,, \tag{3.38}$$

*il cui valore è*

$$\omega_0 = \frac{2}{1+\sqrt{1-\rho^2(J)}} \,. \tag{3.39}$$

Infine se $0 < \omega < 1$ il metodo si dice di *sottorilassamento* mentre se $1 < \omega < 2$ il metodo si dice di *sovrarilassamento*. Facciamo inoltre notare, che quando $\omega = 0$, $H(\omega) = I$ e $\rho(H(\omega)) = 1$.

Esempio 21. Sia

$$A = (a_{i,j}) = \begin{cases} 2 & i = j \\ -1 & |i-j| = 1 \\ 0 & altrimenti \end{cases}$$

Quando $n = 4$, si verifica che $\rho(J) \approx 0.901$ ed essendo tridiagonale simmetrica possiamo determinare $\omega_0$, ottenendo il valore $\omega_0 \approx 1.395$ e $\rho(H(\omega_0)) \approx 0.395$.

Il grafico dell'andamento del raggio spettrale al variare di $\omega$, relativamente all' Esempio 21 ma con $n = 10$, è visibile in Figura 3.1.

In generale, per determinare $\omega_0$, nel caso di una matrice tridiagonale simmetrica, diagonalmente dominante, possiamo avvalerci del codice `SOROmegaZero.m` in Appendice C.

Per comprendere meglio la "filosofia" del metodo SOR, suggeriamo di seguito un paio di esercizi dei quali si chiede di scriversi i corrispondenti M-files.

Esercizio 22. *Si consideri il sistema lineare*

$$\begin{pmatrix} 4 & 0 & 1 & 1 \\ 0 & 4 & 0 & 1 \\ 1 & 0 & 4 & 0 \\ 1 & 1 & 0 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \,.$$

*Si risolva il sistema con il metodo iterativo di Gauss-Seidel a partire dalla soluzione iniziale $(0,0,0,0)$ con precisone di $1.0\mathsf{e}-6$. Si determini inoltre il fattore ottimale di rilassamento per il metodo SOR. Scrivere un M-file che assolva a dette richieste, calcolando anche il numero di iterazioni effettuate.*

*Facoltativo: Determinare la soluzione con il metodo SOR con il fattore ottimo di (sovra)rilassamento.*

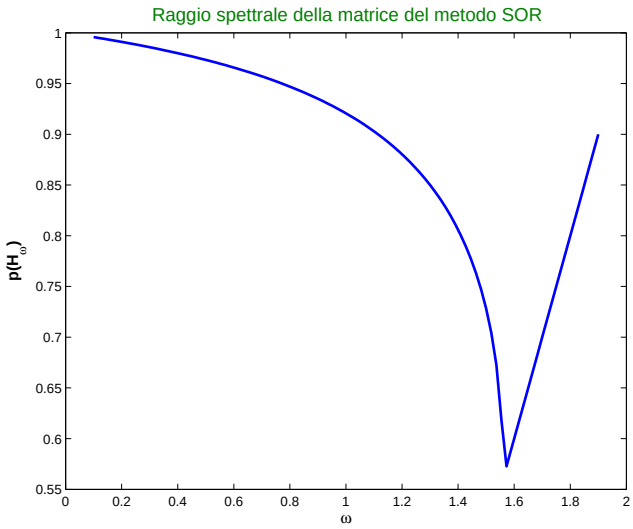


**Figura** 3.1: Raggio spettrale di $H(\omega)$ di dimensione $n = 10$, ottenuto usando la funzione `SOROmegaZero.m`. Il valore ottimale calcolato è $\omega_0 = 1.5727$.

ESERCIZIO 23. *Si consideri il sistema lineare*

$$\begin{pmatrix} 7 & 4 & -7 \\ 4 & 5 & -3 \\ -7 & -3 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 6 \\ -2 \end{pmatrix} .$$

*Si determini sperimentalmente il fattore ottimale di rilassamento per il metodo SOR nel seguente modo a partire dal vettore iniziale* $(0, 0, 0)$. *In pratica si scelgano alcuni* $0 < \omega_i < 2$ *e per ognuno di essi si eseguano 10-15 iterazioni. Quell'* $\omega_i$ *che stabilizza le soluzioni è da considerarsi quello "ottimale" (si osservi che la soluzione del sistema è il vettore (1,1,1)). Perció per sapere quale* $\omega$ *segliere si suggerisce di calcolare per ogni i* `norm([1; 1; 1]-x(`$\omega_i$`),inf)`, *dove* `x(`$\omega_i$`)` *è la soluzione dopo 10-15 iterazioni dell' SOR con* $\omega_i$.

*Facoltativo: Sia A decomposta al solito come* $A = D - B - C$ *e ricordando che la matrice di iterazione del metodo SOR è:*

$$H_\omega = (D - \omega\, B)^{-1}[(1 - \omega)\, D + \omega\, C]$$

*si disegni la funzione* $r : ]0, 2[ \to \mathbb{R}$, *tale che* $r(\omega) = \rho(H_\omega)$. *Verificare quindi se il valore empirico scelto è "vicino" al valore "ottimale" teorico.*

*Scrivere un M-file che assolva a dette richieste, calcolando anche il numero di iterazioni effettuate.*

## 3.7 Sistemi sovra e sottodeterminati

Quando parliamo di *sistemi sovradeterminati* pensiamo a sistemi lineari del tipo $A\,x = b$ con matrice $m \times n$, $m > n$ (ovvero più equazioni che incognite). Se $m < n$ il sistema si dice *sottodeterminato*.

In generale un sistema sovradeterminato non ha soluzione, pertanto si cerca la *"migliore"* soluzione nel senso che ora chiariremo. Dato $b \in \mathbb{R}^m$, diremo che $x^* \in \mathbb{R}^n$ è la migliore soluzione del sistema sovradeterminato, in quanto *minimizza* la norma 2 del residuo $r = b - Ax$. Detto altrimenti,

$$\Phi(x^*) = \|b - Ax^*\|_2^2 \leq \min_{x \in \mathbb{R}^n} \|b - Ax\|_2^2 = \min_{x \in \mathbb{R}^n} \Phi(x)\,. \tag{3.40}$$

**Definizione 16.** *Il vettore $x^*$, quando esiste, si dice* `soluzione ai minimi quadrati` *del sistema $Ax = b$.*

La soluzione ai minimi quadrati è caratterizzata dal seguente teorema.

**Teorema 8.** *Sia $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Se $x^*$ soddisfa l'equazione*

$$A^T(b - Ax^*) = 0 \tag{3.41}$$

*allora per ogni $y \in \mathbb{R}^n$ si ha*

$$\|b - Ax^*\|_2 \leq \|b - Ay\|_2\,. \tag{3.42}$$

**Dim.** Indico con $r_{x^*} = b - Ax^*$ e $r_y = b - Ay$. Ora,

$$r_y = b - Ax^* + Ax^* - Ay = r_{x^*} + A(x^* - y)\,,$$

da cui

$$\begin{aligned} r_y^T r_y &= (r_{x^*} + A(x^* - y))^T\,(r_{x^*} + A(x^* - y)) \\ &= r_{x^*}^T r_{x^*} + r_{x^*}^T A(x^* - y) + (x^* - y)^T A^T r_{x^*} + (x^* - y)^T A^T A(x^* - y)\,. \end{aligned}$$

Pertanto, usando la (3.41), otteniamo

$$\|r_y\|_2^2 = \|r_{x^*}\|_2^2 + \|A(x^* - y)\|_2^2 \geq \|r_{x^*}\|_2^2\,.$$

Questo conclude la dimostrazione. □

Seguono due interessanti osservazioni.

1. Dalla (3.41) segue che per ogni vettore $z \in \mathbb{R}^n$
$$(Az)^T(b - Ax) = 0\,,$$
ovvero il residuo è ortogonale alla soluzione $x$ ai minimi quadrati. Detto altrimenti, il vettore $z$ sta $\mathrm{rg}(A) = \{y \in \mathbb{R}^m,\ y = Ax,\ \forall x \in \mathbb{R}^n\}$.

2. Sempre dalla (3.41), segue che la soluzione $x^*$ ai minimi quadrati è soluzione delle `equazioni normali`
$$(A^T A)x^* = A^T b\,. \tag{3.43}$$

Circa il sistema (3.43), sapendo che la matrice $A$ ha rango $r = \min\{m, n\}$, se ha rango pieno allora è *non singolare* e $B = A^T A$ è simmetrica, definita positiva. Infatti, vale il seguente risultato.

**Teorema 9.** *La matrice $A^T A$ è* `non singolare` *se e solo se le colonne di $A$ sono linearmente indipendenti.*

**Dim.** Se le colonne di $A$ sono linearmente indipendenti, preso $x \neq 0$, $Ax \neq 0$, avremo
$$x^T(A^T A)x = (Ax)^T(Ax) = \|Ax\|_2^2 > 0\,.$$
Quindi $A^T A$ è definita positiva e $\det(A^T A) > 0$ ovvero $A^T A$ è non singolare.

Se le colonne sono linearmente dipendenti, allora $\forall\, z \neq 0$, $Az = 0$ ma anche $A^T Az = 0$ che implica che $A^T A$ è singolare. □

Sotto le ipotesi di questo teorema, allora esiste un'unica soluzione del sistema (nel senso dei minimi quadrati) e il corrispondente sistema si può risolvere mediante la fattorizzazione di Cholesky di $B$.

**Approfondimenti**. A causa degli immancabili errori d'arrotondamento il calcolo di $A^T A$ può introdurre la perdita di cifre significative con il risultato che $A^T A$ non è più definita positiva. In alternativa invece della fattorizzazione di Cholesky si usa la fattorizzazione $QR$ di A.

**Proposizione 5.** *Ogni matrice $A \in \mathbb{R}^{m\times n}$, $m \geq n$ si può scrivere unicamente come $A = QR$ con $Q$ ortogonale quadrata di ordine $m$ e $R \in \mathbb{R}^{m\times n}$ triangolare superiore con le righe di indice $k > n + 1$ tutte nulle.*

### 3.7.1 Fattorizzazione QR di matrici

La fattorizzazione QR di una matrice si può realizzare tramite, ma non solo, *trasformazioni ortogonali di Householder*. Si tratta di *matrici di riflessione*. Infatti, dalla Figura 3.2, è chiaro che il vettore $\mathbf{x}'$, riflesso di $\mathbf{x}$ rispetto all'iperpiano $\pi$, si ottiene come
$$\mathbf{x}' = \mathbf{x} - 2\,\mathbf{v}^T\,\mathbf{x}\,\mathbf{v}$$

dove $\mathbf{x}$ è un versore ortogonale a $\pi$. Pertanto, se indichiamo con $Q_v$ la matrice della trasformazione, che dipende dalla scelta di $\mathbf{v}$, avremo

$$Q_v = I - 2\,\mathbf{v}\,\mathbf{v}^T\ . \tag{3.44}$$

La matrice $Q_v$ si chiama *matrice di Householder*. È facile provare che $Q_v$ soddisfa alle due

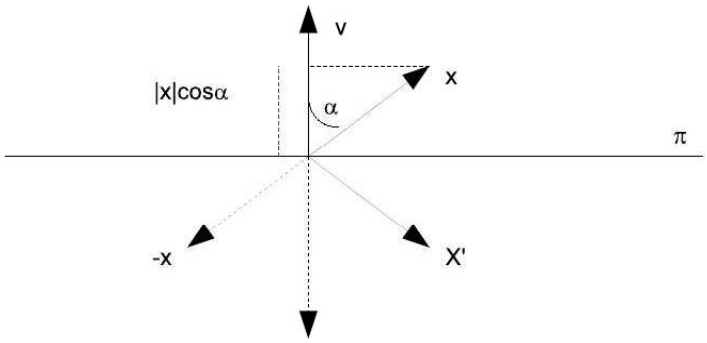


**Figura** 3.2: Riflessione di vettore $\mathbf{x}$ rispetto all'iperpiano $\pi$

seguenti proprietà

- $Q_v$ è simmetrica.
- $Q_v$ è ortogonale.

Si deduce quindi che $Q_v^2 = I$, ovvero che è una matrice involutiva.

La trasformazione di Householder può essere usata per riflettere un vettore in modo tale che tutte le sue coordinate, eccetto una, siano zero. Per semplicità di notazione, traslasciamo l'indice $v$ nella definzione di $Q_v$ e scriveremo $Q$, per indicare la matrice di Houselholder. Detto $\mathbf{x}$ un generico vettore di lunghezza $|\alpha|$ (per questioni di stabilità si può assumere che $\alpha$ abbia lo stesso segno di $x_1$), detto $\mathbf{e}_1 = (1, 0, \ldots, 0)^T$, la matrice $Q_v$ si può allora costruire come segue

1. $\mathbf{u} = \mathbf{x} - \alpha\,\mathbf{e}_1$
2. $\mathbf{v} = \frac{\mathbf{u}}{\|\mathbf{u}\|_2}$
3. $Q = I - 2\,\mathbf{v}\,\mathbf{v}^T$.

Pertanto $Q\,\mathbf{x} = (\alpha, 0, \ldots, 0)^T$.

Questo modo di procedere, si può applicare ad una generica matrice rettangolare $A$, $m\times n$ allo scopo di trasformarla in forma triangolare superiore. Al primo passo, costruiremo $Q_1$ di Houselholder usando la prima colonna di $A$ cosicché

$$Q_1 A = \begin{bmatrix} \alpha_1 & & \\ 0 & & \\ \vdots & A' & \\ 0 & & \end{bmatrix}.$$

Questa modifica può essere ripetuta per la $A$ mediante una matrice di Housholder $Q_2$. Si noti che $Q_2$ più piccola della $Q_1$. Poiché vogliamo che sia reale per operare su $Q_1\,A$ invece di $A'$ abbiamo bisogno di espandere questa nella parte superiore sinistra, riempiendola di 1, o in generale:

$$Q_k = \begin{bmatrix} I_k & 0 \\ 0 & Q_k' \end{bmatrix}.$$

Dopo $p$ iterazioni, $p = \min m-1, n$ avremo

$$R = Q_p Q_{p-1} \cdots Q_1\, A$$

e $Q = Q_1 \cdots Q_p$ è una matrice ortogonale (perchè prodotto di matrici ortogonali). In definitiva $A = QR$ rappresenta la *fattorizzazione* $QR$ di $A$.

$\diamond\diamond$

Se viene usata la fattorizzazione $QR$ di $A$, la soluzione ai minimi quadrati di $A$ si può scrivere come

$$x^* = \tilde{R}^{-1}\tilde{Q}^T b\,,$$

dove $\tilde{R} \in \mathbb{R}^{n\times n}$, $\tilde{Q} \in \mathbb{R}^{m\times n}$ con $\tilde{R} = R(1:n, 1:n)$ e $\tilde{Q} = Q(1:m, 1:n)$ e $\tilde{R}$ non singolare.

Esempio 22. Dati tre punti A,B,C sopra il livello del mare. Per misurare le rispettive altezze sopra il mare, calcoliamo le altezze $h_1, h_2, h_3$ tra altri punti D,E,F ed i punti A,B,C, nonchè le altezze $h_4, h_5, h_6$ tra i punti AB, BC e AC rispettivamente. I valori trovati sono

$$h_1 = 1,\ h_2 = 2,\ h_3 = 3\,;$$

$$h_4 = 1,\ h_5 = 2,\ h_6 = 1\,.$$

Pertanto, ciascuna misurazione da origine ad un sistema lineare che rappresenta la relazione tra le altezze dei punti A,B,C, che indichiamo con $x_A$, $x_B$, $x_C$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_A \\ x_B \\ x_C \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 1 \\ 2 \\ 1 \end{pmatrix} .$$

Le equazioni normali sono

$$\begin{pmatrix} -3 & -1 & -1 \\ -1 & 3 & -1 \\ -1 & -1 & 3 \end{pmatrix} \begin{pmatrix} x_A \\ x_B \\ x_C \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 6 \end{pmatrix} .$$

Risolvendo, ad esempio con la fattorizzazione di Cholesky (ma anche MEG va bene ugualmente) troviamo la soluzione

$$x_A = 3,\ x_B = \frac{7}{4},\ x_C = \frac{5}{4} ,$$

con residuo

$$b - A\mathbf{x} = \begin{pmatrix} -\frac{1}{4}, & \frac{1}{4}, & 0, & \frac{2}{4}, & \frac{3}{4}, & -\frac{3}{4} \end{pmatrix}^T$$

che è ortogonale alle colonne di $A$.

## 3.8 Soluzione di sistemi non lineari con il metodo di Newton

Un sistema non-lineare di $n$ funzioni in $n$ incognite, si può scrivere come il sistema

$$\begin{cases} f_1(x_1, \ldots, x_n) = 0 \\ f_2(x_1, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, \ldots, x_n) = 0 \end{cases} \tag{3.45}$$

dove $f_i : \mathbb{R}^n \to \mathbb{R},\ i = 1, ..., n$ sono funzioni non lineari.

Posto $\mathbf{f} = (f_1, \ldots, f_n)^T$, $\mathbf{x} = (x_1, \ldots, x_n)^T$ e indicato con $\mathbf{0}$ lo zero di $\mathbb{R}^n$, il sistema (3.45) può riscriversi compattamente come $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. Inoltre, se indichiamo con

$$J_{\mathbf{f}}(\mathbf{x}) = \left( \frac{\partial f_i}{\partial x_j} \right)_{i,j=1}^{n}$$

la *matrice jacobiana*, allora possiamo risolvere il predetto sistema con il *metodo di Newton*, che formuleremo come segue

$$\text{risolvi} \quad J_{\mathbf{f}}(\mathbf{x}^{(k)})\delta\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)}),\ \ k = 0, 1, ... \tag{3.46}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \delta\mathbf{x}^{(k)} . \tag{3.47}$$

Il metodo consiste nel risolvere ad ogni passo il sistema lineare (3.46) con matrice del sistema che è la matrice jacobiana.

Due semplici sistemi non lineari

ESEMPIO 23.

$$\begin{cases} x_1^2 + x_2^2 = 0 \\ \mathrm{e}^{x_1} + \mathrm{e}^{x_2} = \log(x_3) \\ x_1 x_2 x_3 = 5 \end{cases}$$

ESEMPIO 24.

$$\begin{cases} x_1^2 + x_2^2 = 1 \\ \sin\left(\frac{\pi x_1}{2}\right) + x_2^3 = 0 \end{cases}$$

Per implementare in Matlab/Octave il metodo di Newton avremo bisogno di una soluzione iniziale $\mathbf{x}_0$, due funzioni `fun` e `jfun` che definiscono la funzione $\mathbf{f}$ e la matrice jacobiana $J_{\mathbf{f}}$, rispettivamente. Come test d'arresto, come al solito, cicleremo finchè

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \leq tol\|\mathbf{x}^{(k)}\| \ \vee \ k > kmax \ .$$

## 3.9 Esercizi proposti

ESERCIZIO 24. `(Laboratorio del 23/01/06)`. *Si consideri la matrice*

$$A = \begin{pmatrix} 1 & \alpha & \alpha \\ \alpha & 1 & \alpha \\ \alpha & \alpha & 1 \end{pmatrix}$$

*Provare graficamente, nel piano $(\alpha, \rho(\alpha))$, che se $\frac{1}{2} \leq \alpha < 1$ il metodo di Gauss-Seidel è convergente mentre quello di Jacobi non lo è.*

*Sia ora $\alpha = \frac{2}{3}$ e $\mathbf{b} = [1\ -1\ 3]'$. Risolvere il sistema $Ax = b$ con Gauss-Seidel: calcolando anche il numero di iterazioni.*

*Trovare la soluzione anche con SOR con $\omega \in [1.2, 1.8]$. Come varia il numero di iterazioni al variare di $\omega$?*

ESERCIZIO 25. *Dato $n \geq 2$ si considerino le matrici $A_k, k = 1, ..., n-1$ di dimensione $n$ definite come segue:*

$$\begin{cases} a_{i,i}^k = n \\ a_{i,j}^k = -1 & |i-j| = k \\ 0 & \textit{altrimenti} \end{cases} .$$

*Sia inoltre* $\mathbf{b} =$ `ones(n,1)`.

*Si risolvano gli $n-1$ sistemi lineari*

$$A_k \mathbf{x} = \mathbf{b}$$

*con il metodo iterativo di Jacobi. La M-function che implementa il metodo di Jacobi dovrà restituire la soluzione, il numero di iterazioni e il massimo autovalore in modulo della matrice di iterazione di Jacobi.*

*Provare servendosi di alcuni grafici, che all'aumentare di k (la grandezza della banda) il numero di iterazioni decresce fino a che $k =$* `floor`$((n+1)/2)$ *per poi stabilizzarsi. Perchè?*

`Facoltativo`*: sia ora* $\mathbf{b} = 1 : n$ *e* $k = n-1$*. Risolvere il sistema* $A_{n-1}\mathbf{x} = \mathbf{b}$ *sia con Jacobi che con Gauss-Seidel. Quale dei due metodi è più veloce?*

ESERCIZIO 26. *Data la matrice tridiagonale*

$$A = \begin{bmatrix} d & -1 & & \\ -1 & d & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & d \end{bmatrix},$$

*con $d \geq 2$, si risolva il sistema lineare $A\,x = b$ con $b$ tale che $x = (1, \ldots, 1)^T$. Valutando la norma euclidea della differenza tra due iterate successive, ovvero*

$$\delta_{k+1} = \|x^{(k+1)} - x^{(k)}\|$$

*nei casi $d = 2, 3$ presentiamo in tabella alcune di tali differenze*

| $d=2$ | | $d=3$ | |
|---|---|---|---|
| $\vdots$ | | $\vdots$ | |
| 456 | $7.2754e-3$ | 16 | $1.0229e-4$ |
| 457 | $7.2616e-3$ | 17 | $6.5117e-5$ |
| 458 | $7.2477e-3$ | 18 | $4.1563e-5$ |
| 459 | $7.2340e-3$ | 19 | $2.6593e-5$ |

- *Si stimi in norma 2, il numero di iterazioni $m$ necessarie nei casi $d = 2$ e $d = 3$ affinchè la differenza $\|\mathbf{x}^{k+m} - \mathbf{x}^{k+m-1}\| \leq 1.e-9$ partendo da $k = 458$ e $k = 18$, rispettivamente. (*`Sugg.`*: È noto che*

  $$\delta_{k+1} \leq C_k \delta_k$$

  *con $C_k$ la norma 2 della matrice d'iterazione al passo $k$. Usando i valori tabulati, dapprima si determini un' approssimazione di $C_k$ nei due casi $d = 2$ e $d = 3$ e quindi iterando ... )*

- *Scrivere inoltre un programma Matlab che risolve il sistema precedente usando il metodo di Jacobi, prendendo come dati in ingresso d, n, b, tol, senza allocare la matrice A e la matrice di iterazione di Jacobi, partendo da* $\mathbf{x}^0 = 0$. *Lo si applichi nel caso* $d = 3$, $n = 10$, $\mathbf{b}$=`ones(n,1)` *e* $tol = 1.e-9$.

ESERCIZIO 27. (`Appello del 26/9/05`). *Dati i sistemi lineari* $A_1x = b$ *e* $A_2y = b$ *con*

$$A_1 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 13 & 18 & 23 & 28 \\ 3 & 18 & 50 & 62 & 74 \\ 4 & 23 & 62 & 126 & 148 \\ 5 & 28 & 74 & 148 & 255 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 3 & 4 & 5 & 6 \\ 0 & 0 & 5 & 6 & 7 \\ 0 & 0 & 0 & 7 & 8 \\ 0 & 0 & 0 & 0 & 9 \end{bmatrix}$$

*e il termine noto*

$$b = [\texttt{15}, \texttt{18}, \texttt{18}, \texttt{15}, \texttt{9}]^{\texttt{T}}.$$

1. *Risolvere i due sistemi con un opportuno metodo diretto.*

2. *Sia* $\delta b = \texttt{rand}(\texttt{5},\texttt{1}) * \texttt{1.e} - \texttt{3}$ *una perturbazione del vettore b. Si risolvano ora i due sistemi perturbati* $A_1x = b + \delta b$ *e* $A_2y = b + \delta b$. *Confrontando le nuove soluzioni con quelle ottenute al punto precedente, dire quale sistema risulta meglio condizionato analizzando la quantità*

$$\frac{\|E_x^{rel}\|_2}{\|E_b^{rel}\|_2},$$

*dove* $E^{rel}$ *indica l'errore relativo.*

*Osservazione.* $A_1 = A_2^T A_2$, *quindi i numeri di condizionamento in norma 2 di* $A_1$ *e* $A_2$ *saranno legati ..... Inoltre questo garantisce che* $A_1$ *è simmetrica definita positiva per cui sarà possibile applicare il metodo....*

ESERCIZIO 28. (`Laboratorio del 23/01/06`). *Si consideri la matrice*

$$A = \begin{pmatrix} -0.5 & \alpha & 0.5 \\ 0.5 & -0.5 & \alpha \\ \alpha & 0.5 & -0.5 \end{pmatrix}, \quad \alpha \in \mathbb{R}. \tag{3.48}$$

1. *Individuare un intervallo* $I_\alpha$ *di valori di* $\alpha$ *in cui la matrice d'iterazione del metodo di Gauss-Seidel è convergente. (Sugg.: calcolare al variare di* $\alpha$ *l' autovalore di modulo maggiore usando* `eig` *e quindi .... ).*

2. *Preso* $\alpha^* \in I_\alpha$, *risolvere il sistema* $Ax = b$ *con b tale che* $x = [1, 1, 1]^T$, *con il metodo di Gauss-Seidel con* $tol = 1.e-6$, *determinando anche il numero di iterazioni e come test di arresto sul residuo* $r_k = b - Ax_k$, *ovvero iterando finchè* $\|r_k\| > tol\|b\|$.

ESERCIZIO 29. *Data la matrice*

$$A = \texttt{diag(ones(7,1)*10)} + \texttt{diag(ones(6,1)*3,+1)} + \texttt{diag(ones(6,1)*3,-1)}$$

*e il termine noto*

$$b = [\texttt{1 2 3 4 5 6 7}]^{\texttt{T}}.$$

1. *Dire perchè convergono i metodi di Jacobi e Gauss-Seidel.*
2. *Fissata la tolleranza $\tau = 1.e-9$ e sia $P$ la matrice di iterazione tale che $\|P\| < 1$, allora risolvendo*
$$\frac{\|P\|^k}{1-\|P\|}\|x^1 - x^0\| < \tau$$
*possiamo calcolare a priori il numero di iterazioni $k$ necessarie per ottenere una soluzione a meno di $\tau$. Partendo dalla soluzione iniaziale* `x0=0` *e usando la norma infinito, $\|\ \|_\infty$ determinare $k$ sia per il metodo di Jacobi che Gauss-Seidel.*
3. *Verificare sperimentalmente i risultati ottenuti applicando i metodi di Jacobi e Gauss-Seidel, rispettivamente, alla soluzione del sistema $A\,x = b$.*

ESERCIZIO 30. `(Appello del 29/3/07)`. *Data la matrice* `A=pascal(5)` *di ordine $n = 5$,*

1. *Determinare $M = \max_{1\le i\le n}\{\lambda_i\}$, $m = \min_{1\le i\le n}\{\lambda_i\}$. Usare $tol = 1.e-6$.*
2. *Studiare il metodo iterativo dipendente dal parametro reale $\theta \in [0, 1/2]$*
$$x^{(k+1)} = (I - \theta A)x^{(k)} + \theta\, b\,, \quad k \ge 0\,. \tag{3.49}$$
*Si chiede di verificare graficamente per quali valori di $\theta$ il metodo converge.*
3. *Sia $\theta^* = \min\{0 \le \theta \le 1/2\}$ per cui il metodo iterativo converge. Siano $b \in \mathbb{R}^n$ tale che* `x=ones(n,1)` *e* `x0=zeros(n,1)`. *Risolvere quindi il sistema $Ax = b$ con il metodo iterativo (3.49).*

ESERCIZIO 31. `(Appello del 20/7/07)`. *Si consideri la matrice $A \in \mathbb{R}^{10\times 10}$*

$$A = \begin{bmatrix} 5 & -1 & & & \\ -1 & 5 & -1 & & \\ & -1 & 5 & -1 & \\ & & \ddots & \ddots & \\ & & & -1 & 5 \end{bmatrix},$$

*e il vettore $b =$* `ones(10,1)`.

1. *Si dica (senza fare calcoli) se i metodi iterativi di Jacobi e Gauss-Seidel convergono alla soluzione del sistema $Ax = b$.*

2. *Si consideri ora il metodo iterativo di* `Richardson stazionario` *per la soluzione di* $Ax = b$*:*

$$x^{(k+1)} = (I - \alpha\, A)x^{(k)} + \alpha\, b$$

*dove* $\alpha \in [0.01, 0.3]$ *è un parametro di accelerazione. Si chiede di stimare il parametro ottimale* $\alpha^*$ *(quello per il cui il metodo di Richardson converge più rapidamente).*

3. *Produrre un grafico comparativo dell'errore assoluto, in funzione del numero delle iterazioni fatte, ottenuto con i metodi di Jacobi, Gauss-Seidel e Richardson stazionario con parametro* $\alpha^*$*. Usare:* `x0` $=$ `zeros`$(10, 1)$, `tol` $=$ `1.e` $- 6$, `nmax` $=$ `100`.

ESERCIZIO 32. *Si consideri la matrice* $A =$ `pentadiag`$(-1, -1, 10, -1, -1) \in \mathbb{R}^{10\times 10}$ *che possiamo decomporre in* $A = M+D+N$ *con* $D =$ `diag`$([9, 9, ...., 9])$*,* $M =$ `pentadiag`$(-1, -1, 1, 0, 0)$ *e* $N = A - M - D$*.*

*Si considerino i seguenti schemi iterativi*

1. $(M + D)x^{(k+1)} = -Nx^{(k)} + q\,,$
2. $Dx^{(k+1)} = -(M + N)x^{(k)} + q\,,$
3. $(M + N)x^{(k+1)} = -Dx^{(k)} + q\,.$

*Dire quali di essi è convergente analizzando il raggio spettrale delle matrici d'iterazione.*

*Sia poi* `q=1:10`*. Si calcoli la soluzione del sistema* $A\,x = q$ *con uno dei metodi convergenti, a partire dalla soluzione* $x^{(0)} =$`[ones(2,1);zeros(8,1)]` *a meno di tol* $= 1.e - 6$*.*

ESERCIZIO 33. `(Appello del 11/9/07)`*. Si consideri la matrice* $A =$ `pentadiag`$(-1, -1, \alpha, -1, -1) \in \mathbb{R}^{10\times 10}$*,* $\alpha \in [0.5, 1.5]$ *che possiamo decomporre in* $A = M + D + N$ *con* $D =$ `diag`$([\alpha - 1, ...., \alpha - 1])$*,* $M =$ `pentadiag`$(-1, -1, 1, 0, 0)$ *e* $N = A - M - D$*.*

1. *Per quale valore* $\alpha^*$ *il metodo iterativo* $(M + N)x^{(k+1)} = -Dx^{(k)} + q$ *risulta essere convergente più velocemente?*
2. *Sia poi* `q=1:10`*. Si calcoli la soluzione del sistema* $A\,x = q$ *a partire dalla soluzione* $x^{(0)} =$`[ones(5,1);zeros(5,1)]` *a meno di tol* $= 1.e - 6$*.*

# Capitolo 4

# Calcolo di autovalori di matrici

## 4.1 Autovalori di matrici

Iniziamo introducendo alcune utili definizioni.

**Definizione 17.** *Data una matrice quadrata $A \in \mathbb{R}^{n\times n}$, si chiama* `autovalore` *di $A$, quel numero reale o complesso $\lambda$ tale che per ogni vettore $x \neq 0$ soddisfa l'equazione*

$$Ax = \lambda x\,. \tag{4.1}$$

Il vettore $x$ viene detto `autovettore` associato all'autovalore $\lambda$. Osserviamo che l'autovettore $x$ non è unico. Infatti, se $\alpha \in \mathbb{R}$ è un qualsiasi numero reale non nullo, allora il vettore $y = \alpha x$ è ancora un autovettore associato all'autovalore $\lambda$.

Se l'autovettore $x$ è noto, il corrispondente autovalore si può determinare usando il `quoziente di Rayleigh`

$$\lambda = \frac{x^T A x}{x^T x}\,. \tag{4.2}$$

Dalla definizione precedente, segue che $\lambda$ è autovalore di $A$ se è una radice del `polinomio caratteristico`

$$p_A(\lambda) = \det(A - \lambda\, I)\ .$$

Infatti, l'equazione (4.1) è equivalente a

$$(A - \lambda I)x = 0$$

ma essendo $x \neq 0$ essa sarà soddisfatta se e solo se la matrice $A - \lambda I$ risulta essere singolare. Inoltre, il polinomio caratterisco associato ad una matrice $A$ di ordine $n$, ha $n$ radici reali e/o complesse. Se $\lambda \in \mathbb{C}$ è autovalore di $A$, anche $\bar{\lambda}$ è un autovalore complesso di $A$.

Premettiamo due utili risultati circa gli autovalori di matrici con struttura. Il primo ci ricorda che le trasfomazioni per similitudine conservano gli autovalori. Mentre il secondo ci ricorda che le matrici simmetriche hanno autovalori reali.

**Proposizione 6.** *Matrici simili hanno gli stessi autovalori*

**Dim.** Siano $A$ e $B$ simili, ovvero $P^{-1}AP = B$, con $P$ invertibile. Ora, se $\lambda$ è autovalore di $A$ e $x \neq 0$ è l'autovettore associato, allora

$$BP^{-1}x = P^{-1}Ax = \lambda P^{-1}x\,.$$

Quindi $\lambda$ è autovalore di $B$ con autovettore associato $P^{-1}x$. □

**Proposizione 7.** *Matrici simmetriche hanno autovalori reali.*

**Dim.** Dapprima osserviamo che per il coniugato del polinomio caratteristico di $A$ valgono le identità

$$\overline{\det(A-\lambda I)} = \det(A-\lambda I)^* = \det(A^* - \bar{\lambda} I)$$

dove $A^*$ è la matrice trasposta e coniugata di $A$. Si deduce allora che gli autovalori di $A^*$ sono i coniugati di quelli di $A$. Valendo le identità

$$x^*A^*x = \bar{\lambda}\|x\|_2^2\,, \quad x^TAx = \lambda\|x\|_2^2\,,$$

si conclude che $\bar{\lambda} = \lambda$ ovvero se e solo se $\lambda \in \mathbb{R}$. □

◇◇

**Definizione 18.** *Una matrice $A \in \mathbb{R}^{n\times n}$ è diagonalizzabile, se esiste una matrice $U \in \mathbb{R}^{n\times n}$ tale che*

$$U^{-1}AU = \Lambda\,, \tag{4.3}$$

*con $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ e $U$ che ha per colonne gli $n$ autovettori di $A$ (che formano una base per $\mathbb{R}^n$).*

Nel caso di matrice rettangolare non parliamo di autovalori ma di `valori singolari`. Vale il seguente risultato noto come `decomposizione` ai valori singolari (o SVD).

**Teorema 10.** *Sia $A \in \mathbb{R}^{m\times n}$. Allora esistono due matrici ortogonali $U \in \mathbb{R}^{m\times m}$ e $V \in \mathbb{R}^{n\times n}$ tali che*

$$U^TAV = \Sigma\,, \tag{4.4}$$

*con $\Sigma = diag(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}$, $p = \min\{m, n\}$, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.*

I numeri $\sigma_i$ sono i `valori singolari` di $A$ di cui parleremo più oltre nella sezione dedicata alla soluzione del problema del "data fitting" e "decomposizione SVD" (vedi Sezione 5.9).

Infine c'è un interessante risultato di *localizzazione* degli autovalori di una matrice.

**Definizione 19.** *Data $A$ di ordine $n$, i cerchi*

$$C_i^{(r)} = \{z \in \mathbb{C} : |z - a_{i,i}| \leq \sum_{j=1, j\neq i}^{n} |a_{i,j}| \}, i = 1, \ldots, n \tag{4.5}$$

$$C_i^{(c)} = \{z \in \mathbb{C} : |z - a_{i,i}| \leq \sum_{j=1, j\neq i}^{n} |a_{j,i}| \}, i = 1, \ldots, n \tag{4.6}$$

*sono cerchi* `riga` *e* `colonna` *e sono detti i* `cerchi di Gerschgorin` *associati alla matrice $A$.*

Vale il seguente Teorema (che non dimostreremo).

**Teorema 11.** *Le seguenti affermazioni sono vere.*

1. *Ogni autovalore di $A$ appartiene alla regione del piano complesso*

$$R = \bigcup_{i=1}^{n} C_i^{(r)} .$$

*Ogni autovalore di $A$ appartiene alla regione del piano complesso*

$$C = \bigcup_{i=1}^{n} C_i^{(c)} .$$

*Pertanto essi appartengono anche all'intersezione $R \cap C$.*

2. *Ogni componente di $R$ o $C$, ovvero ogni componente connessa massimale formata da $R_i$ cerchi o $C_j$ cerchi, contiene tanti autovalori di $A$ quanti sono i cerchi che la compongono (contando anche la molteplicità di ogni autovalore e di ogni cerchio).*

A supporto di questo Teorema facciamo un esempio tratto da [18, pag. 89].

Esempio 25. Sia

$$A = \begin{pmatrix} 4 & -1 & 1 & 0 & 0 \\ 1 & 3 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 1 & 8 \end{pmatrix}$$

i cui autovalori sono $\lambda_1 = 5 + \sqrt{10}$, $\lambda_2 = \lambda_3 = 3$, $\lambda_4 = 2$ e $\lambda_5 = 5 - \sqrt{10}$. I cerchi riga sono

$$\begin{aligned} R_1 &= \{z : |z - 4| \leq 2\}, \\ R_2 &= \{z : |z - 3| \leq 2\}, \\ R_3 &= \{z : |z - 1| \leq 1\}, \\ R_4 &= \{z : |z - 2| \leq 1\}, \\ R_5 &= \{z : |z - 8| \leq 1\} . \end{aligned}$$

quelli colonna sono

$$\begin{aligned} C_1 &= \{z : |z-4| \leq 1\}, \\ C_2 &= \{z : |z-3| \leq 2\}, \\ C_3 &= \{z : |z-1| \leq 2\}, \\ C_4 &= \{z : |z-2| \leq 1\}, \\ C_5 &= \{z : |z-8| \leq 1\} . \end{aligned}$$

I grafici dei corrispondenti cerchi di Gerschgorin sono riprodotti in Figura 4.1. È facile osservare che gli autovalori stanno nell'insieme

$$R_2 \cup R_3 \cup R_4 \cup R_5$$

poiché $R_2 = C_2,\ R_4 \subset R_2;\ C_1, C_4 \subset C_2$ e $R_5 = C_5$.

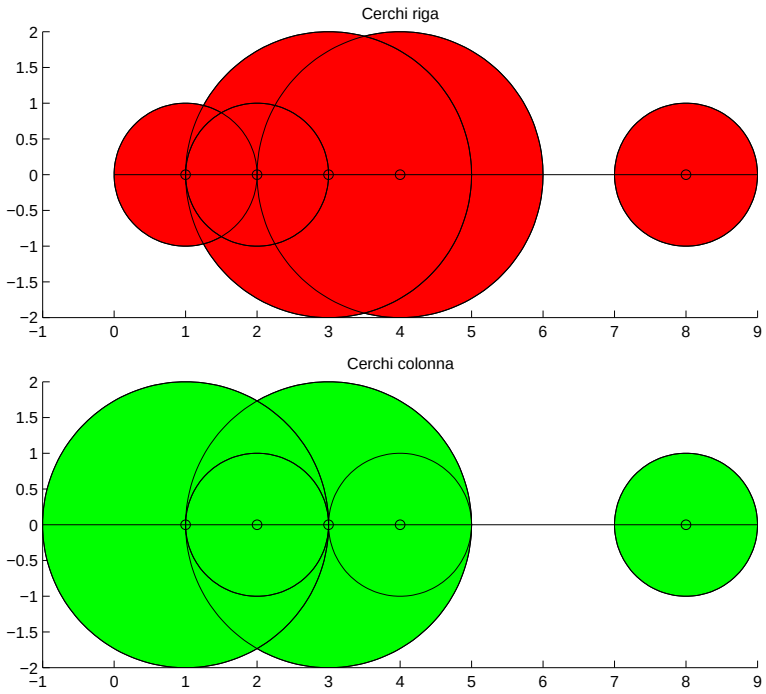


**Figura** 4.1: Cerchi di Gerschgorin della matrice $A$ dell' Esempio 25: sopra i cerchi riga e sotto quelli colonna.

L'Esempio 25, ci suggerisce che se $A$ è simmetrica allora le regioni $R$ e $C$ del Teorema 11 coincidono ed essendo gli autovalori di matrici simmetriche reali, la loro intersezione è formata dall'unione di intervalli dell'asse reale.

**Proposizione 8.** *Se $A$ è diagonalmente dominante in senso stretto allora è non singolare.*

**Dim.** La dimostrazione è ora facilitata dalla conoscenza dei cerchi di Gerschgorin. Infatti, se $A$ è diagonalmente dominante vale la disguaglianza

$$\sum_{j=1, j\neq i}^{n} \left| \frac{a_{i,j}}{a_{i,i}} \right| < 1 \, .$$

Ciò implica che

$$\frac{|z - a_{i,i}|}{|a_{i,i}|} = \sum_{j=1, j\neq i}^{n} \left| \frac{a_{i,j}}{a_{i,i}} \right| < 1 \, , \tag{4.7}$$

da cui

$$|z - a_{i,i}| < |a_{i,i}| \, .$$

La matrice $A$ ha quindi cerchi di Gerschgorin che non passano mai per l'origine e pertanto non potrà mai avere autovalori nulli. □

La funzione `CerchiGerschgorin.m`, in Appendice C, consente di costruire e plottare i cerchi di Gerschgorin.

Infine, ricordando che $\rho(A) \leq \|A\|$ per ogni norma indotta, una sovrastima dell'autovalore di modulo più grande è appunto $\|A\|$.

Le domande più frequenti quando si ha a che fare con problemi di autovalori sono le seguenti.

1. Quali autovalori desideriamo conoscere? Il più grande in modulo o il più piccolo in modulo? E cosa si può dire dei corrispondenti autovettori?
2. E se volessimo determinare tutti gli autovalori e autovettori?
3. La struttura della matrice che ruolo gioca nei metodi di calcolo?

## 4.2 Il metodo delle potenze

Il *metodo delle potenze* permette di determinare l'autovalore di modulo massimo.

Supponiamo che gli autovalori di $A$ possano essere ordinati come segue:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \, ,$$

con $\lambda_1$ ben distinto dai rimanenti autovalori. Sia $\mathbf{x}_1$ l'autovettore corrispondente a $\lambda_1$.

Se gli autovettori di A sono linearmente indipendenti, $\lambda_1$ e $\mathbf{x}_1$ si possono determinare come segue

1. Dato il vettore iniziale $\mathbf{x}^{(0)}$, poniamo $\mathbf{y}^{(0)} = \mathbf{x}^{(0)}/\|\mathbf{x}^{(0)}\|$.

2. Per $k = 1, 2, ...$ calcolo

$$\mathbf{x}^{(k)} = A\mathbf{y}^{(k-1)}, \;\; \mathbf{y}^{(k)} = \frac{\mathbf{x}^{(k)}}{\|\mathbf{x}^{(k)}\|}, \;\; \lambda^{(k)} = (\mathbf{y}^{(k)})^T \, A\, \mathbf{y}^{(k)} \; .$$

La procedura si arresta in corrispondenza al primo indice $k$ tale che $|\lambda^{(k)} - \lambda^{(k-1)}| < \epsilon|\lambda^{(k)}|$.

Il predetto metodo costruisce *due successioni convergenti*

$$\lim_{k\to\infty} \mathbf{y}^{(k)} = \alpha \mathbf{x}_1 \, , \quad \lim_{k\to\infty} \lambda^{(k)} = \lambda_1 \; .$$

Perchè si chiama *metodo delle potenze*? Basta osservare che $\mathbf{y}^{(k)} = r^{(k)} A^k \mathbf{y}^{(0)}$, cioè appaiono le potenze della matrice $A$, con

$$r^{(k)} = \prod_{i=1}^{k} \frac{1}{\|\mathbf{x}^{(i)}\|} \, .$$

Infatti,

- $y^{(1)} = \dfrac{Ay^{(0)}}{\|x^{(1)}\|}$;
- $y^{(2)} = \dfrac{Ay^{(1)}}{\|x^{(2)}\|} = \dfrac{A^2 y^{(0)}}{\|x^{(1)}\|\|x^{(2)}\|}$;
- $y^{(3)} = \dfrac{Ay^{(2)}}{\|x^{(3)}\|} = \dfrac{A^3 y^{(0)}}{\|x^{(1)}\|\|x^{(2)}\|\|x^{(3)}\|}$;
- ....

La funzione `MetPotenze.m`, in Appendice C, implementa il metodo delle potenze.

### 4.2.1 Convergenza del metodo delle potenze

Gli autovettori $x_1, \ldots, x_n$ sono linearmente indipendenti, cosicché possiamo scrivere $x^{(0)} = \sum_{i=1}^{n} \alpha_1 x_i$ da cui

$$y^{(0)} = \beta^{(0)} \sum_{i=1}^{n} \alpha_i x_i, \;\; \beta^{(0)} = 1/\|x^{(0)}\| \, .$$

(i) Al primo passo

$$\begin{aligned} x^{(1)} &= Ay^{(0)} = \beta^{(0)} A \sum_{i=1}^{n} \alpha_i x_i = \beta^{(0)} \sum_{i=1}^{n} \alpha_i \lambda_i x_i \\ y^{(1)} &= \beta^{(1)} \sum_{i=1}^{n} \alpha_i \lambda_i x_i, \;\; \beta^{(1)} = \left( \|x^{(0)}\| \|x^{(1)}\| \right)^{-1} \end{aligned}$$

(ii) Al passo $k$

$$y^{(k)} = \beta^{(k)} \sum_{i=1}^{n} \alpha_i \lambda_i^k x_i, \;\; \beta^{(k)} = \left( \prod_{i=0}^{k} \|x^{(i)}\| \right)^{-1}$$

da cui

$$y^{(k)} = \lambda_1^k \beta^{(k)} \left( \alpha_1 x_1 + \sum_{i=2}^{n} \alpha_i \left( \frac{\lambda_i}{\lambda_1} \right)^k x_i \right),$$

Poiché $\left|\frac{\lambda_i}{\lambda_1}\right| < 1$, $i = 2, ..., n$ allora per $\lim_{k\to\infty} y^{(k)} = x_1$ o meglio converge alla direzione dell'autovettore $x_1$ purchè $\alpha_1 \neq 0$. Nel caso in cui $\alpha_1 = 0$ e $|\lambda_2| > |\lambda_3|$, il processo *dovrebbe* convergere verso $\lambda_2$. Però a causa degli (inevitabili) errori di arrotondamento si ha $\alpha_1 \neq 0$ e quindi il metodo converge ancora a $\lambda_1$.

Quando però $|\lambda_2| \approx |\lambda_1|$, la successione $\{\lambda_1^{(k)}\}$ converge verso $\lambda_1$ molto lentamente e in tal caso il metodo viene usato come stima iniziale per il *metodo delle potenze inverse* che vedremo oltre.

Inoltre, se la matrice $A$ non è simmetrica, la convergenza all'autovalore di modulo massimo è $\mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|\right)$. Quando $A$ è simmetrica la convergenza raddoppia, ovvero é $\mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^2\right)$.

Concludendo, il rapporto $\left|\frac{\lambda_2}{\lambda_1}\right|$ è importante ai fini della velocità di convergenza del metodo delle potenze. Il prossimo esempio ci fa capire proprio questo fatto.

Esempio 26. Si consideri la matrice

$$A_1 = \begin{pmatrix} -7 & -9 & 9 \\ 11 & 13 & -9 \\ -16 & -16 & 20 \end{pmatrix}$$

i cui autovalori sono $\lambda_1 = 20, \lambda_2 = 4, \lambda_3 = 2$ con $\left|\frac{\lambda_2}{\lambda_1}\right| \approx 0.2$. Si può provare che partendo dal vettore $x^{(0)} = [1, 1, 1]^T$ con $tol = 1.e-6$, il metodo delle potenze (implementato nella funzione `MetPotenze.m`) converge a $\lambda_1$ in 10 iterazioni.

Se consideriamo invece la matrice

$$A_2 = \begin{pmatrix} -4 & -5 & 4 \\ 14 & 15 & -5 \\ -1 & -1 & 11 \end{pmatrix}$$

che ha autovalori $\lambda_1 = \frac{\sqrt{5}+21}{2}, \lambda_2 = \frac{\sqrt{5}-21}{2}, \lambda_3 = 1$. In tal caso $\left|\frac{\lambda_2}{\lambda_1}\right| \approx 0.81$ con la conseguenza che il metodo, con gli stessi valori di *tol* e $x^{(0)}$, impiega 58 iterazioni per determinare l'autovalore di modulo massimo.

ESERCIZIO 34. *Si consideri, per $\alpha \in \mathbb{R}$, la famiglia di matrici*

$$A = \begin{pmatrix} \alpha & 2 & 3 & 10 \\ 5 & 12 & 10 & 7 \\ 9 & 7 & 6 & 13 \\ 4 & 16 & 18 & 0 \end{pmatrix}$$

*Provare che se $\alpha = 30$ il metodo converge all'autovalore di modulo massimo in 27 iterazioni, mentre se $\alpha = -30$ il metodo richiede ben 1304 iterazioni partendo da $x^{(0)} =$*`ones(4,1)` *con tolleranza $\epsilon = 1.e-10$. Come mai questo fatto?*

*Per verificare l'esattezza dei calcoli, usare la funzione* `eig(A)` *di Matlab/Octave che restituisce tutti gli autovalori di una matrice quadrata A. Come ulteriore controllo determinare anche il residuo $A\,\mathbf{x}_1 - \lambda_1 \mathbf{x}_1$.*

### 4.2.2 Il metodo delle potenze inverse

Per determinare l'autovalore di modulo minimo, basta ricordare che $A^{-1}$ ha autovalori che sono i reciprochi degli autovalori della matrice $A$. Infatti, se $\lambda$ è autovalore di $A$ associato all'autovettore $x$, allora da $Ax = \lambda x$ deduciamo che $\frac{1}{\lambda}x = A^{-1}x$. Ovvero $1/\lambda$ è autovalore di $A^{-1}$ associato al vettore $x$.

Da un punto di vista implementativo, al passo $k$ invece di definire $x^{(k)} = A^{-1}y^{(k-1)}$ risolveremo, con uno dei metodi numerici visti al capitolo precedente, il sistema

$$Ax^{(k)} = y^{(k-1)} .$$

Se fosse nota la fattorizzazione $LU$ o quella di Cholesky (nel caso $A$ sia simmetrica definita positiva), basterà ricordarla ed usarla ad ogni passo $k$.

ESERCIZIO 35. *Si consideri la matrice dell'Esercizio 34, con gli stessi valori del parametro $\alpha$, determinare l'autovalore di modulo minimo $\lambda_n$ mediante il metodo delle potenze inverse (ovvero il metodo delle potenze applicato ad $A^{-1}$). Usare $x^{(0)} =$*`ones(4,1)` *e tolleranza $\epsilon = 1.e-10$.*

### 4.2.3 Il metodo delle potenze inverse con shift

Data una matrice quadrata A $n \times n$, a coefficienti reali, i cui autovalori possono essere ordinati come segue:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \, .$$

Con il *metodo delle potenze con shift* è possibile cercare l' autovalore di $A$, più vicino ad numero $\eta$ fissato. In pratica si tratta di applicare il *metodo delle potenze inverse* per il calcolo dell'autovalore minimo $\lambda_{min}(A_\eta)$ della matrice $A_\eta = A - \eta\, I$. L'autovalore cercato, dipendente da $\eta$, è $\lambda_\eta = \lambda_{min}(A_\eta) + \eta$.

Per individuare un valore $\eta$ da cui partire, si possono costruire i *cerchi di Gerschgorin*, $C_i^{(r)}$ e $C_i^{(c)}$, $i = 1, ..., n$ (vedi (4.5) e (4.6)), associati alle righe e alle colonne di $A$, rispettivamente (vedasi la funzione `CerchiGerschgorin.m`).

Esercizio 36. *Cercare l'/gli autovalore/i di A, con $\alpha = -30$, più vicino/i al numero $\eta = -15$. In pratica si tratta di applicare il metodo delle potenze inverse per il calcolo dell'autovalore minimo della matrice $A_\eta = A - \eta\, I$. Quindi l'autovalore cercato sarà $\lambda_\eta = \lambda_{min}(A_\eta) + \eta$.*

*Come cambia il numero delle iterazioni se prendiamo $\eta = -17$? Prendere $x^{(0)}$ =*`ones(4,1)` *e tolleranza $\epsilon = 1.e-10$.*

### 4.2.4 Metodo delle potenze e metodo di Bernoulli

Dato il polinomio

$$p_n(x) = \sum_{i=0}^{n} a_i x^i, \;\; a_0\, a_n \neq 0 \, ,$$

per determinare la radice $\xi_1$ di modulo massimo si può usare il *metodo di Bernoulli*. Tale metodo consiste nell'applicare il *metodo delle potenze* alla matrice (di Frobenius), detta anche *matrice companion*

$$F = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ 0 & \cdots & & 0 & 1 \\ -\frac{a_0}{a_n} & -\frac{a_1}{a_n} & \cdots & -\frac{a_{n-2}}{a_n} & -\frac{a_{n-1}}{a_n} \end{pmatrix} .$$

Per calcolare tutti i rimanenti autovalori si opera poi per deflazione, ovvero applicando il metodo alle sottomatrici di ordine $n-1$, $n-2$, ..., 1 che si ottengono mediante trasformazioni per similitudine con matrici ortogonali (quali le matrici di Householder).

Facciamo vedere su un semplice esempio come calcolare la radice più grande in modulo, che chiameremo $\xi_1$, e la successiva radice più grande in modulo $\xi_2$.

ESEMPIO 27. Calcolare $\xi_1$ per il polinomio

$$p_6(x) = 13x^6 - 364x^5 + 2912x^4 - 9984x^3 + 16640x^2 - 13312x + 4096 ,$$

usando tolleranza $tol = 1.e-6$.

Calcolare quindi (per deflazione) $\xi_2$ la seconda radice più grande in modulo. Per calcolare $\xi_2$, si suggerisce dapprima di costruire la matrice $P$ di *Householder* tale

$$PFP^T = \begin{pmatrix} \xi_1 & \mathbf{a}^T \\ \mathbf{0} & F_1 \end{pmatrix}$$

cosicchè $Px_1 = e_1$ con $x_1$ autovettore associato a $\xi_1$ (calcolato al passo 1) e $e_1 = (1, 0, ..., 0)^T$. La matrice $P$ si può costruire mediante il seguente codice Matlab:

```
% Costruisco la matrice di Householder P t.c. P*x1=(1,0,...,0)
x12=norm(x1,2); beta=1/(x12*(x12+abs(x1(1)) ));
v(1)=sign(x1(1))*(abs(x1(1))+x12); v(2:n)=x1(2:n);
P=eye(n)-beta*v'*v;   % n = dimensione matrice
```

Pertanto, per calcolare $\xi_2$ si applicherà il metodo delle potenze alla matrice $F_1$.

Confrontare i risultati con la funzione `roots(c)`, con `c` vettore dei coefficienti del polinomio $p_6(x)$, aiutandosi anche con un grafico.

Una possibile implementazione in Matlab/Octave del metodo di Bernoulli, per calcolare tutte le radici della matrice di Frobenius è presentata nella funzione `metBernoulli` in Appendice C.

## 4.3 Il metodo QR

Se si desiderano **tutti** gli autovalori di una matrice, bisogna ricorrere a tecniche che consentono dapprima di ridurre la matrice ad una forma più semplice mediante trasformazioni per similitudine pervenendo a una forma triangolare superiore o diagonale: il calcolo degli autovalori diventa così notevolemente semplificato. Questa è la filosofia delle *trasformazioni* con matrici ortogonali di Householder o Givens. Su tale filosofia si basa infatti il *metodo QR e le sue varianti*.

Il metodo $QR$ fa uso della fattorizzazione $QR$ della matrice $A$. La fattorizzazione $QR$ di $A$ consiste nel premoltiplicare $A$ ad ogni passo $k$ per una *matrice ortogonale*, di Householder

o Givens, cosicchè dopo $n-1$ passi $(U_{n-1}\cdots U_1)A = R$, con $R$ triangolare superiore. La matrice $Q$ richiesta è

$$Q = (U_{n-1}\cdots U_1)^{-1} = (U_{n-1}\cdots U_1)^T .$$

Il metodo *QR per autovalori* si descrive come segue. Sia $A \in \mathbb{R}^{n\times n}$; data $Q^{(0)} \in \mathbb{R}^{n\times n}$ ortogonale (cioè $Q^TQ = I$) e posto $T^{(0)} = (Q^{(0)})^T A Q^{(0)}$, per $k = 1, 2, ...$ finché converge esegui

- mediante la fattorizzazione QR di A (ad esempio usando la funzione `qr` di Matlab/Octave o una propria implementazione), determinare $T^{(k-1)} = Q^{(k)}R^{(k)}$;
- quindi, porre $T^{(k)} = R^{(k)}Q^{(k)} = (Q^{(k)})^T T^{(k-1)} Q^{(k)}$.

Se $A$ ha autovalori *reali e distinti in modulo* il metodo converge ad una matrice triangolare superiore i cui autovalori stanno sulla diagonale principale. Nel caso in cui gli autovalori non soddisfino la predetta condizione, la successione converge verso una matrice con forma triangolare a blocchi, come si vede nel prossimo esempio.

Esempio 28. Data

$$A = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} .$$

Dopo 25 iterazioni del QR, si perviene alla matrice in forma "quasi" triangolare (di Hessenberg superiore)

$$T^{(25)} = \begin{bmatrix} 2 & 1.069 & 0.9258 \\ 0 & -0.5 & 0.866 \\ 0 & -0.866 & -0.5 \end{bmatrix} ,$$

i cui autovalori sono $\lambda_1 = 2$ e $\lambda_{2,3}$ che si determinano dal blocco

$$\begin{bmatrix} -0.5 & 0.866 \\ -0.866 & -0.5 \end{bmatrix} ,$$

che sono complessi coniugati $\lambda_{2,3} = -0.5 \pm 0.866\, i$.

Vale la seguente Proposizione

**Proposizione 9.** *Data $A \in \mathbb{R}^{n\times n}$, esiste $Q \in \mathbb{R}^{n\times n}$ ortogonale e una matrice $B \in \mathbb{R}^{n\times n}$ triangolare superiore a blocchi tale che*

$$B = Q^T A Q = \begin{pmatrix} R_{1,1} & R_{1,2} & \cdots & R_{1,m} \\ & R_{2,2} & \cdots & R_{2,m} \\ & & \ddots & \vdots \\ 0 & & & R_{m,m} \end{pmatrix} \tag{4.8}$$

*dove* $R_{i,i},\ i = 1, ..., m$ *è un blocco* $2 \times 2$ *con autovalori complessi coniugati o un blocco* $1 \times 1$ *nel caso l'autovalore sia un numero reale. La somma delle dimensioni dei blocchi* $R_{i,i}$, $i = 1, ..., m$ *è pari ad* $n$. *Inoltre*

$$Q = \lim_{k\to\infty} \left[ Q^{(0)} \ \cdots \ Q^{(k)} \right] ,$$

*dove* $Q^{(k)}$ *è la* $k$*-esima matrice ortogonale generata al passo* $k$ *della fattorizzazione* `QR` *di* $A$.

La matrice con blocchi $R_{i,j}$ viene detta la *decomposizione reale di Schur* di $A$.

Poiché il metodo ha lo scopo di annullare gli elementi della parte triangolare inferiore sotto la diagonale principale partendo dall'elemento in basso a sinistra, un possibile *test* di arresto è che al passo $k$

$$\sum_{i=1}^{n-1} \left| T_{i+1,i}^{(k)} \right| < \epsilon$$

ovvero che tutti gli elementi della sottodiagonale siano in modulo minori di una prescelta tolleranza $\epsilon$ (vedasi più sotto).

In Appendice C la funzione `MetQR` implementa il metodo QR per la ricerca di autovalori, mentre la funzione `ConvergenzaQR` verifica se il metodo QR converge, controllando che gli elementi della sottodiagonale siano tutti in modulo minori di una fissata tolleranza.

### 4.3.1 Il metodo QR con shift

Dalla Proposizione 9, abbiamo visto che il metodo QR converge, nel caso generale, verso una matrice triangolare superiore a blocchi. Risolvendo quindi il problema degli autovalori sui blocchi, si avranno tutti gli autovalori di $A$ (vedasi l'Esempio 28).

Il metodo QR ha velocità che dipende dai rapporti $|\lambda_i/\lambda_j|,\ i > j$ ed in particolare dal numero $\max_{1\le i\le n-1} \left| \frac{\lambda_{i+1}}{\lambda_i} \right|$, più questo numero è vicino ad 1 e più lenta sarà la convergenza, pociché $A$ ha ha autovalori vicini in modulo. In questi casi, conviene applicare la tecnica di traslazione dello spettro, o *tecnica dello shift*, che serve anche ad accelerare la convergenza.

Il *metodo QR con singolo shift* consiste nella seguente iterazione: data l'approssimazione $\mu \in \mathbb{R}$ di un autovalore $\lambda$ di $A$, consideriamo la forma di Hessenberg di $A$, ovvero $T^{(0)} = (Q^{(0)})^T A Q^{(0)}$ (in Matlab/Octave si ottiene usando la funzione `hess.m`). Quindi

- per $k = 1, 2, ...$ mediante la fattorizzazione QR di $A$ (usare la funzione Matlab/Octave `qr`), fattorizzare $T^{(k-1)}$ come segue: $T^{(k-1)} - \mu I = Q^{(k)} R^{(k)}$;
- porre $T^{(k)} = R^{(k)} Q^{(k)} + \mu I$.

Osserviamo che $Q^{(k)}T^{(k)} = Q^{(k)}R^{(k)}Q^{(k)} + \mu Q^{(k)} = T^{(k-1)}Q^{(k)}$, ovvero $T^{(k)}$ e $T^{(k-1)}$ sono simili e pertanto hanno gli stessi autovalori.

L'effetto dello shift sulla convergenza è il seguente. Supponiamo che $A$ abbia autovalori che possiamo ordinare

$$|\lambda_1 - \mu| \geq |\lambda_2 - \mu| \geq \cdots |\lambda_n - \mu|$$

si può provare che per $1 < i \leq n$, $t_{i,i-1}^{(k)} \to 0$ con velocità proporzionale a

$$\left| \frac{\lambda_i - \mu}{\lambda_{i-1} - \mu} \right|^k$$

estendendo così lo stesso risultato di convergenza visto per il metodo QR anche al QR con shift. Questo suggerisce, di scegliere per $\mu$ un valore che approssima $\lambda_n$ cosicchè

$$|\lambda_n - \mu| < |\lambda_i - \mu| \,, \quad i = 1, \ldots, n-1 \,.$$

Per tale scelta di $\mu$, l'elemento $t_{n,n-1}^{(k)}$, generato dalla precedente iterazione, tende rapidamente a zero al crescere di $k$. Se per caso $\mu$ fosse un autovalore della matrice $T^{(k)}$, e anche di A, $t_{n,n-1}^{(k)} = 0$ e $t_{n,n}^{(k)} = \mu$. Questo suggerisce in ultima istanza di scegliere

$$\mu = t_{n,n}^{(k)} \,.$$

Con questa scelta, si dimostra, la convergenza del metodo è quadratica, nel senso che se

$$\frac{t_{n,n-1}^{(k)}}{\|T^{(0)}\|_2} = \alpha_k < 1, \quad \text{per qualche } k \geq 0$$

allora

$$\frac{t_{n,n-1}^{(k+1)}}{\|T^{(0)}\|_2} = \mathcal{O}(\alpha_k^2) \,.$$

Nel caso di matrice simmetrica, si dimostra (cf. [12]) che la convergenza è cubica.

Di questo fatto possiamo tenere conto durante l'esecuzione del metodo *QR con singolo shift*, controllando il valore di $|t_{n,n-1}^{(k)}|$ e ponendolo uguale a zero se risulta

$$|t_{n,n-1}^{(k)}| < \epsilon \left( |t_{n-1,n-1}^{(k)}| + |t_{n,n}^{(k)}| \right) \quad \epsilon \approx \text{eps.} \tag{4.9}$$

Questo sarà il test da usare nell'implementazione del metodo QR con shift. Se, la matrice $A$ è in forma di Hessenberg superiore, l'azzeramento di $t_{n,n-1}^{(k)}$ implica che $t_{n,n}^{(k)}$ sarà una buona approssimazione di $\lambda_n$. Quindi, noto $\lambda_n$ la ricerca dei rimanenti autovalori si farà sulla matrice $T^{(k)}(1:n-1, 1:n-1)$, riducendo di volta in volta la dimensione del problema fino a determinare **tutti** gli autovalori di $A$. In pratica una strategia di tipo deflattivo.

Il *metodo QR con singolo shift* è implementato nella funzione `MetQRShift` in Appendice C.

◇◇

ESERCIZIO 37. Calcolare con il metodo QR tutti gli autovalori di `A=[30 1 2 3; 4 15 -4 -2; -1 0 3 5; -3 5 0 -1];`. Determinare anche il numero di iterazioni fatte.

ESERCIZIO 38. Si consideri la matrice $A$ (tratta da [20, pag. 178])

$$A = \begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix},$$

i cui autovalori (arrotondati a due decimali ) sono $\lambda_1 = 65$, $\lambda_{2,3} = \pm 21.28$ e $\lambda_{4,5} = \pm 13.13$. Calcolare, se possibile, tutti gli autovalori sia con il metodo QR che con il metodo QR con shift.

### 4.3.2 Autovalori di matrici simmetriche

Quando la matrice $A \in \mathbb{R}^{n\times n}$ è *tridiagonale simmetrica* o *simmetrica*, per la ricerca dei corrispondenti autovalori si usano il *metodo delle successioni di Sturm* e il *metodo di Jacobi*, rispettivamente.

## 4.4 Il metodo delle successioni di Sturm

Sia $A \in \mathbb{R}^{n\times n}$ tridiagonale simmetrica. Indichiamo con **d** e **b** i vettori diagonale e extradiagonali (sopra e sotto la diagonale principale) di dimensioni $n$ e $n-1$, rispettivamente.

Sia $A_i$ il minore principale di ordine $i$ di $A$. Posto $p_0(x) = 1$, indichiamo con $p_i(x) = \det(A_i - xI_i)$. Si ha

$$\begin{aligned} p_1(x) &= d_1 - x; \\ p_i(x) &= (d_i - x)\, p_{i-1}(x) - b_{i-1}^2 p_{i-2}(x), \quad i = 2, ..., n. \end{aligned} \tag{4.10}$$

Alla fine $p_n(x) = \det(A - xI)$. La successione $\{p_i(x)\}$ è detta una *successione di Sturm*. Vale il seguente risultato (la cui dimostrazione si trova in [2, pagg. 345-346]).

**Teorema 12.**

1. *$p_0(x)$ non cambia segno;*
2. *se $p_i(x) = 0$ allora $p_{i-1}(x)p_{i+1}(x) < 0$, per $i = 1, 2, \ldots, n-1$; (alternanza degli zeri)*
3. *se $p_n(x) = 0$ allora $p_n'(x)p_{n-1}(x) < 0$ (e quindi $p_n(x)$ ha tutti zeri di molteplicità 1).*

Detto altrimenti, per $i = 2, ..., n$ gli autovalori di $A_{i-1}$ **separano quelli di** $A_i$, ovvero

$$\lambda_i(A_i) < \lambda_{i-1}(A_{i-1}) < \lambda_{i-1}(A_i) < \cdots < \lambda_2(A_i) < \lambda_1(A_{i-1}) < \lambda_1(A_i)\,. \tag{4.11}$$

Inoltre, per ogni reale $\nu$, definiamo

$$S_\nu = \{p_0(\nu), p_1(\nu), \ldots, p_n(\nu)\}\,. \tag{4.12}$$

Allora $s(\nu)$, il numero di cambiamenti di segno in $S_\nu$, indica il *numero di autovalori di $A$ strettamente minori di $\nu$*. Vale inoltre,

**Teorema 13.** *Se $p_i(x),\ i = 0, 1, \ldots, n$ è una successione di Sturm, il numero $s(b) - s(a)$ indica il numero di zeri di $p_n(x)$ appartenenti all'intervallo $[a, b)$.*

Da un punto di vista implementativo, per costruire $A$, noti i vettori **d** e **b**, basta usare il comando Matlab/Octave `A=diag(d)+diag(b,1)+diag(b,-1)`. Quindi si può procedere come segue:

- Scelgo un valore reale $\nu$ e costruisco l'insieme $S_\nu$. Qui bisogna implementare le formule (4.10), ottenendo in output un array che contiene i numeri $p_i(\nu),\ i = 0, 1, ..., n$.
- Determino il numero $s(\nu)$ che mi dirà, grazie alla (4.11), quanti autovalori di $A$ sono minori in senso stretto, di $\nu$.
- Esiste un metodo, detto *metodo di Givens*, per determinare tutti gli autovalori di una matrice $A$ tridiagonale simmetrica. Poniamo $b_0 = b_n = 0$ allora l'intervallo $I = [\alpha, \beta]$ con

  $$\alpha = \min_{1\le i\le n}(d_i - (|b_i| + |b_{i-1}|)), \quad \beta = \max_{1\le i\le n}(d_i + (|b_i| + |b_{i-1}|))\,, \tag{4.13}$$

  conterrà tutti gli autovalori di $A$ (infatti $\alpha$ e $\beta$ indicano gli estremi dell'intervallo di Gerschgorin).

  Una volta determinato $I = [\alpha, \beta]$, per determinare $\lambda_i$, l'$i$-esimo autovalore di $A$, mediante il metodo di bisezione si opera come segue: si costruiscono le successioni $a^{(i)}$ e $b^{(i)}$ con $a^{(0)} = \alpha$, $b^{(0)} = \beta$; quindi si calcola $c^{(0)} = (a^{(0)} + b^{(0)})/2$, **grazie alla proprietà (4.11)**, se $s(c^{(0)}) > n - i$ allora $b^{(1)} = c^{(0)}$ altrimenti $a^{(1)} = c^{(0)}$. Si continua finchè ad un passo $k^*$, $|b^{(k^*)} - a^{(k^*)}| \le \text{tol}(|a^{(k^*)}| + |b^{(k^*)}|)$.

  Procederemo poi in modo sistematico per calcolare tutti gli altri autovalori.

ESERCIZIO 39. *Data la matrice tridiagonale avente diagonale principale il vettore* `d=ones(1,n)` *e sopra e sotto diagonali il vettore* `b=-2*ones(1,n-1)`. *Con* $n = 4$, *calcolarne tutti gli autovalori usando il metodo di Givens delle successioni di Sturm.*

Per facilitare l'implemetazione presentiamo la funzione `succSturm.m` che costruisce le successione di Sturm e i suoi cambiamenti di segno.

```
function [p,cs]=succSturm(d,b,x)
%-----------------------------------------------
% Calcolo della successione di Sturm 'p' in x
% a partire dai vettori d e b
% e dei corrispondenti cambiamenti di segno 'cs'
%-----------------------------------------------
n=length(d); p(1)=1; p(2)=d(1)-x; for i=2:n,
    p(i+1)=(d(i)-x)*p(i)-b(i-1)^2*p(i-1);
end
cs=0; %contatore cambi di segno
s=0;  %contatore dei segni costanti
for i=2:length(p),
    if(p(i)*p(i-1)<=0),
        cs=cs+1;
    end
    if(p(i)==0),
        s=s+1;
    end
end
    cs=cs-s;
return
```

## 4.5 Il metodo di Jacobi

Il metodo, come detto, si applica a *matrici simmetriche*. Genera una successione di matrici $A^{(k)}$ ortogonalmente simili ad $A$ e convergenti verso una matrice diagonale i cui elementi sono gli autovalori di $A$.

Si parte da $A^{(0)} = A$. Per $k = 1, 2, \ldots$ si fissa una coppia di interi $p$ e $q$ con $1 \leq p < q \leq n$ e si costruisce

$$A^{(k)} = (G_{pq})^T A^{(k-1)} G_{pq} \tag{4.14}$$

(ortogonalmente simile ad A) cosicchè

$$a_{i,j}^{(k)} = 0, \quad \text{se} \ \ (i,j) = (p,q)\,.$$

La matrice $G_{pq}$ è la matrice ortogonale di rotazione di Givens definita come segue

$$G_{pq} = \begin{bmatrix} 1 & & & & & & 0 \\ & \ddots & & & & & \\ & & \cos(\theta) & & \sin(\theta) & & \\ & & & \ddots & & & \\ & & -\sin(\theta) & & \cos(\theta) & & \\ & & & & & \ddots & \\ 0 & & & & & & 1 \end{bmatrix}.$$

Siano $c = \cos(\theta)$ e $s = \sin(\theta)$. Allora, gli elementi di $A^{(k)}$ che variano rispetto a quelli di $A^{(k-1)}$ per effetto della trasformazione (4.14) si ottengono risolvendo il sistema

$$\begin{bmatrix} a_{pp}^{(k)} & a_{pq}^{(k)} \\ a_{pq}^{(k)} & a_{qq}^{(k)} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp}^{(k-1)} & a_{pq}^{(k-1)} \\ a_{pq}^{(k-1)} & a_{qq}^{(k-1)} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}. \tag{4.15}$$

Il nostro scopo è trovare l'angolo $\theta$ che consente al passo $k$ di annullare gli elementi extradiagonali di $A^{(k-1)}$ interessati dalla trasformazione. Ora, se $a_{pq}^{(k-1)} = 0$, allora $c = 1$ e $s = 0$. Se invece $a_{pq}^{(k-1)} \neq 0$, posto $t = s/c = \tan(\theta)$, il sistema (4.15) equivale alla soluzione dell'equazione

$$t^2 + 2\eta t - 1 = 0, \quad \text{con } \eta = \frac{a_{qq}^{(k-1)} - a_{pp}^{(k-1)}}{2a_{pq}^{(k-1)}}. \tag{4.16}$$

Nell'equazione precedente, se $\eta \geq 0$ si sceglie la radice $t = 1/(\eta + \sqrt{1+\eta^2})$ altrimenti $t = -1/(-\eta + \sqrt{1+\eta^2})$. Quindi $c$ e $s$ risultano

$$c = \frac{1}{\sqrt{1+t^2}}, \quad s = ct.$$

La convergenza del metodo si verifica calcolando la seguente quantità, valida per una generica matrice $M$

$$\Phi(M) = \left( \sum_{i,j=1, i\neq j}^{n} m_{ij}^2 \right)^{1/2} = \left( \|M\|_F^2 - \sum_{i=1}^{n} m_{ii}^2 \right)^{1/2}. \tag{4.17}$$

Il metodo garantisce che $\Phi(A^{(k)}) \leq \Phi(A^{(k-1)}), \ \forall k$. Infatti

$$\begin{aligned} \Phi(A^{(k)}) &= \sum_{i,j=1}^{n} a_{i,j}^2 - \sum_{i=1}^{n} a_{i,i}^2 - 2|a_{p,q}| \\ &= \Phi(A^{(k-1)}) - 2|a_{p,q}| \\ &< \Phi(A^{(k-1)}). \end{aligned} \tag{4.18}$$

Una strategia ottimale di scelta degli indici $p, q$ tale che $\Phi(A^{(k)}) \leq \Phi(A^{(k-1)})$ sia sempre verificata, è quella di scegliere l'elemento di $A^{(k-1)}$ tale che

$$|a_{p,q}^{(k-1)}| = \max_{i \neq j} |a_{i,j}^{(k-1)}| \,.$$

Vale infatti il seguente

**Teorema 14.** *La successione $A^{(k)}$ generata con il metodo di Jacobi è convergente verso una matrice diagonale e si ha* $\lim_{k\to\infty} \Phi(A^{(k)}) = 0$.

**Dim.** Essendo $a_{p,q}$ un elemento non principale di massimo modulo di $A^{(k)}$, risulta

$$a_{p,q}^2 \geq \frac{\Phi(A^{(k-1)})}{n(n-1)} \,.$$

Dalla (4.18),

$$\Phi(A^{(k)}) \leq \Phi(A^{(k-1)}) - 2\frac{\Phi(A^{(k-1)})}{n(n-1)} = \alpha\Phi(A^{(k-1)})$$

con $\alpha = 1 - \frac{2}{n(n-1)} < 1,\ n \geq 2$. Continuando,

$$\Phi(A^{(k)}) \leq \alpha^{k-1}\Phi(A^{(1)})$$

da cui la conclusione. □

Una M-function che implementa il metodo di Jacobi, data $A$ e una tolleranza *tol* e che restituisce la matrice diagonale $D$ degli autovalori, il numero di iterazioni effettuate e la quantità $\Phi(\cdot)$, è la funzione `symJacobi` in Appendice C.

## 4.6 Esercizi proposti

ESERCIZIO 40. *Data la matrice* `simmetrica`.

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 & 0 \\ 1 & 3 & 2 & 0 & 0 \\ 0 & 2 & -1 & -4 & 0 \\ 0 & 0 & -4 & 6 & 2 \\ 0 & 0 & 0 & 2 & 5 \end{pmatrix}$$

1. *Localizzare, mediante i cerchi di Gerschgorin, gli autovalori di $A$ e dare alcune stime "a priori".*

2. *Determinare tutti gli autovalori con il metodo più idoneo per la struttura della matrice.*

ESERCIZIO 41. `(Appello del 23/3/05)`. *Data la matrice simmetrica.*

$$A = \begin{pmatrix} 4 & 1 & 1 & 0 \\ 1 & 3 & 2 & 3 \\ 1 & 2 & -1 & -2 \\ 0 & 3 & -2 & 5 \end{pmatrix}$$

1. *mediante il metodo delle potenze determinare l'autovalore di modulo massimo $M$, con precisione $tol = 1.e-6$;*
2. *mediante il metodo delle potenze inverse determinare l'autovalore di modulo minimo $m$, con precisione $tol = 1.e-6$;*
3. *si determinino infine gli altri due autovalori (interni a $[-M, M]$.)*

ESERCIZIO 42. *Data la matrice $A$ di ordine $n = 5$,*

$$A = \begin{pmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{pmatrix},$$

*i cui autovalori (arrotondati a due decimali ) sono $\lambda_1 = 65$, $\lambda_{2,3} = \pm 21.28$ e $\lambda_{4,5} = \pm 13.13$. Calcolare tutti gli autovalori con il* **metodo QR con shift**. *Costruire una tabella che riporti i valori della sequenza*

$$\rho_k = 1 + \frac{1}{\log \eta_k} \log \frac{|t^{(k)}_{n,n-1}|}{|t^{(k-1)}_{n,n-1}|}, \quad k = 1, 2, ....$$

*con $\eta_k = |t^{(k)}_{n,n-1}|/\|T^{(0)}\|_2$, $T^{(0)} = (Q^{(0)})^T A Q^{(0)}$ e $Q^{(0)}$ (la prima delle matrici ortogonali usati nella fattorizzazione $QR$ di $A$) che faccia vedere che la convergenza del metodo è quadratica.*

ESERCIZIO 43. *Si considerino le matrici*

$$A_1 = \begin{pmatrix} -7 & -9 & 9 \\ 11 & 13 & -9 \\ -16 & -16 & 20 \end{pmatrix} \quad A_2 = \begin{pmatrix} -4 & -5 & 4 \\ 14 & 15 & -5 \\ -1 & -1 & 11 \end{pmatrix} \tag{4.19}$$

*entrambe con autovalori reali e distinti.*

1. *Calcolare tutti gli autovalori di $A_1$ e $A_2$, mediante il metodo delle inverse con shift (usare $tol = 1.e-6$).*

2. *Si dica perchè il metodo delle potenze per il calcolo dell'autovalore di modulo massimo ci impiega di più nel caso della matrice $A_2$?*

ESERCIZIO 44. `(Appello del 23/3/06)`. *Si consideri la matrice*

$$A = \begin{pmatrix} \frac{1}{3} & \frac{2}{3} & 2 & 3 \\ 1 & 0 & -1 & 2 \\ 0 & 0 & -\frac{5}{3} & -\frac{2}{3} \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{4.20}$$

1. *Perchè il metodo delle potenze non funziona per il calcolo dell'autovalore di modulo massimo?*

2. *Calcolare l'autovalore di modulo massimo e il corrispondente autovettore con il metodo delle potenze con shift.*

ESERCIZIO 45. `(Laboratorio del 6/2/07 )` *Data la matrice di Hilbert di ordine 4 (in Matlab* `hilb(4)`*), i cui autovalori (arrotondati) sono* $\lambda_1 = 1.5002$, $\lambda_2 = 0.1691$, $\lambda_3 = 0.0067$, $\lambda_4 = 0.0001$. *Calcolare detti autovalori usando il metodo di Jacobi con una tolleranza* `tol=1.e-15`. *In quante iterazioni converge il metodo? Calcolare anche ad ogni iterazione la quantità* $\Phi(A^{(k)})$ *per verificare che decresce. (Sugg. Usare la funzione* `symJacobi.m` *(implementare le M-functions* `calcoloCeS` *e* `calcoloProdottoGtDG`*)).*

# Capitolo 5

# Interpolazione e approssimazione

Nelle applicazioni si conoscono solitamente dati provenienti da campionamenti di una funzione $f$ sui valori $x_i, i = 0, \ldots, n$, ovvero $(x_i, f(x_i))$ oppure dati sparsi provenienti di misurazioni $(x_i, y_i), i = 0, ..., n$. Il problema dell'interpolazione consiste nel trovare una funzione $\tilde{f}$ tale da soddisfare le `condizioni d'interpolazione`

$$\tilde{f}(x_i) = f(x_i),\ \textit{oppure}\ ,\ \tilde{f}(x_i) = y_i\ . \tag{5.1}$$

A seconda della forma di $\tilde{f}$ parleremo di interpolazione

- `polinomiale`: $\tilde{f}(x) = a_0 + a_1 x + \cdots + a_n x^n := p_n(x)$ ;
- `polinomiale a tratti`: in tal caso su ognuno degli intervallini $I_k = [x_k, x_{k+1}],\ k = 0, ..., n-1$ $\tilde{f}$ coincide con un polinomio. Nel caso questo polinomio sia una `spline` parleremo d'interpolazione spline;
- `trigonometrica`: $\tilde{f}(x) = a_{-M}\mathrm{e}^{-iMx} + \cdots + a_M \mathrm{e}^{iMx}$, ove $M = n/2$ se $n$ pari ovvero $M = (n-1)/2$ se $n$ è dispari;
- `razionale`: in tal caso $\tilde{f}(x) = \frac{p_n(x)}{q_m(x)}$, con $p_n$ e $q_m$ polinomi di grado $n$ e $m$ rispettivamente.

## 5.1 Interpolazione polinomiale

Il seguente Teorema dice che il problema dell'interpolazione polinomiale ha un'unica soluzione se i punti d'interpolazione $x_i$, su cui costruiremo il polinomio interpolante, sono `distinti`.

**Teorema 15.** *Dati $n+1$ punti $(x_i, y_i)$, $i = 0, ..., n$ con $x_i \neq x_j$, $i \neq j$, esiste un unico polinomio di grado $n$, $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n$ per cui*

$$p_n(x_i) = y_i,\ i = 0, ..., n \tag{5.2}$$

**Dim.** Le condizioni (5.2) sono equivalenti al sistema lineare

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

la cui matrice $A$ altro non è che la matrice di `Vandermonde`. Si dimostra (vedi anche [7]) che

$$\det(A) = \prod_{i,j=0,\, i<j}^{n} (x_i - x_j)\ ,$$

Tale determinante è diverso zero perchè $x_i \neq x_j, i \neq j$. Pertanto la soluzione esiste ed è unica. □

ESEMPIO 29. Si consideri la `funzione di Runge`

$$g(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5] \tag{5.3}$$

Sugli $n+1$ punti equispaziati $x_i = -5 + ih,\ i = 0, 1, ..., n, \quad h = 10/n$ si costruisca il polinomio d'interpolazione di grado $n$, $p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$. Si tratta di risolvere il sistema lineare

$$V\ \mathbf{a}\ =\ \mathbf{y}$$

con $\mathbf{a} = (a_0, a_1, ..., a_n)^T$, $y = (g(x_0), g(x_1), ..., g(x_n))^T$ e $V$ la matrice di Vandermonde. Ad esempio se $n = 3$, $x_0 = -5$, $x_1 = -5 + 10/3 \approx -1.667$, $x_2 = -5 + 20/3 \approx 1.667, x_3 = 5$ il sistema diventa

$$V = \begin{pmatrix} 1 & -5 & 25 & -125 \\ 1 & -1.667 & 2.779 & -4.63 \\ 1 & 1.667 & 2.779 & 4.63 \\ 1 & 5 & 25 & 125 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{26} \approx 0.04 \\ 0.26 \\ 0.26 \\ 0.04 \end{pmatrix}.$$

◇◇

Il concetto di *condizioni d'interpolazione* può essere generalizzato, come vediamo nel prossimo esempio, considerando non solo i valori della funzione nei nodi ma altri valori.

ESEMPIO 30. Si consideri la funzione $f(x) = \dfrac{20}{1+x^2} - 5\,\mathrm{e}^x$ ristretta all'intervallo $[0,1]$. Determinare l'unico polinomio di secondo grado, $p_2(x) = a_0 + a_1 x + a_2 x^2$ tale che

$$p_2(0) = f(0),\ p_2(1) = f(1),\ \int_0^1 p_2(x)dx = \int_0^1 f(x)dx\ .$$

Si tratta di risolvere il sistema lineare con matrice non singolare

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & \frac{1}{2} & \frac{1}{3} \end{pmatrix}$$

e termine noto

$$b = \begin{pmatrix} 15 \\ 5(2-\mathrm{e}) \\ \\ \displaystyle\int_0^1 f(t)dt \end{pmatrix}$$

L'integrale definito è facilmente calcolabile analiticamente ottenendo $20\arctan(1) - 5(e - 1) = 5\pi - 5\mathrm{e} + 5 \approx 7.1166$. Risolvendo il sistema si trovano i valori dei coefficienti del polinomio. Il sistema lo possiamo risolvere anche numericamente con il MEG o usando la funzione Matlab/Octave "\": otteniamo $a_0 = 15,\ a_1 = -10.118; a_2 = -8.474$. Il grafico della funzione e del polinomio sono visibili in Fig. 5.1.

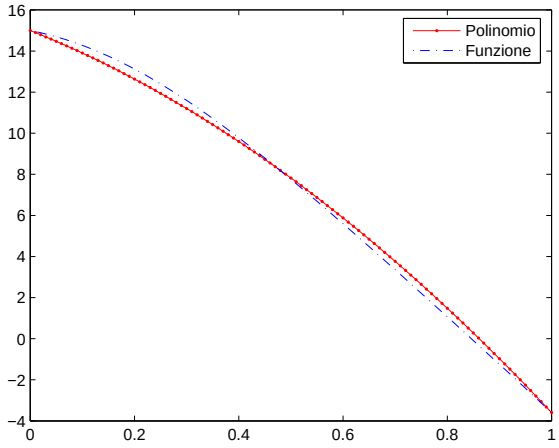


**Figura** 5.1: Funzione e polinomio d'interpolazione dell'Esempio 30

## 5.2 Forma di Lagrange dell'interpolante

Il polinomio d'interpolazione si può esprimere usando la base di *Lagrange*. I *polinomi elementari di Lagrange* sono definibili a partire dai punti d'interpolazione $x_i$ come segue

$$l_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j} \ . \tag{5.4}$$

I polinomi $l_i,\ i = 0, ..., n$ sono polinomi di grado $n$, valgono 1 quando $x = x_i$ e 0 altrimenti, cioè $l_i(x_j) = \delta_{i,j}$ (vedasi Figura 5.2). Pertanto possiamo esprimere il polinomio d'interpolazione $p_n(x)$, costruito sull'insieme $(x_i, f(x_i))$, $i = 0, ..., n$, come

$$p_n(x) = \sum_{i=0}^{n} l_i(x) f(x_i) \ . \tag{5.5}$$

Inoltre vale la seguente identità

$$\sum_{i=0}^{n} l_i(x) = 1 \ ,$$

che altro non è che il polinomio d'interpolazione della funzione $f(x) \equiv 1$.

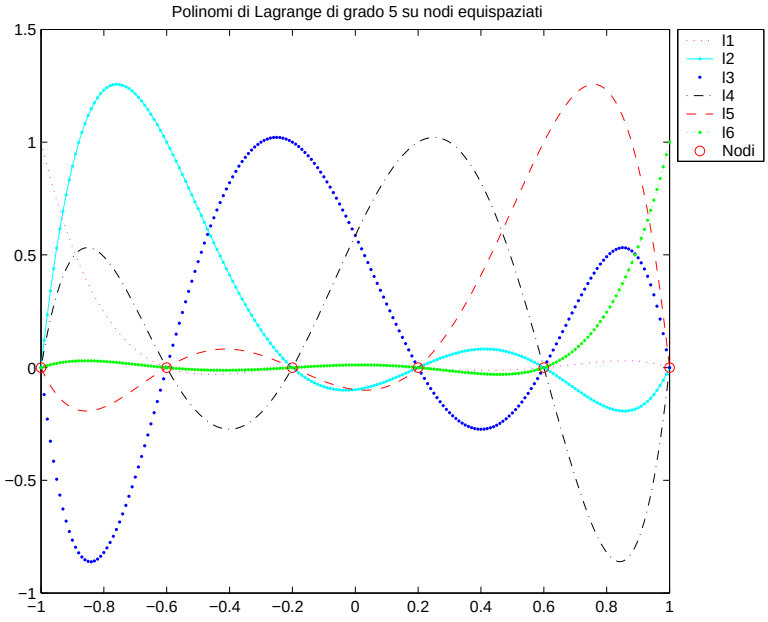


**Figura** 5.2: Grafico di alcuni polinomi elementari di Lagrange.

Posto $\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$ è facile provare che

$$l_i(x) = \frac{\omega_{n+1}(x)}{(x - x_i)\omega'_{n+1}(x_i)} .$$

Pertanto una forma alternativa del polinomio d'interpolazione è (cfr. [1, 3]):

$$p_n(x) = \omega_{n+1}(x) \sum_{i=0}^{n} \frac{f(x_i)}{(x - x_i)\omega'_{n+1}(x_i)} .$$

Tale forma, detta *formula baricentrica*, ha un costo computazionale di ordine $n^2$ invece che ordine $n^3$ come per la forma (5.5). Infatti, sono richiesti $\mathcal{O}(n^2)$ flops per calcolare i *pesi* $w'_{n+1}(x_i)$. Pertanto, noti i pesi, per valutare il polinomio $p_n(x)$, serviranno $\mathcal{O}(n)$ flops.

Infine, come osservazione finale, i polinomi elementari $l_i$, essi valgono 1 in $x_i$ e zero negli altri punti $x_j, j \neq i$, ma possono assumere valore maggiore di 1 in modulo, come si evince anche dalla Figura 5.2.

Esempio 31. Consideriamo il caso $n = 1$. I punti che prendiamo sono $(x_0, f(x_0))$ e $(x_1, f(x_1))$. Il corrispondente sistema di Vandermonde è

$$\begin{pmatrix} 1 & x_0 \\ 1 & x_1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \end{pmatrix} .$$

La matrice inversa è

$$\frac{1}{x_1 - x_0} \begin{pmatrix} x_1 & -x_0 \\ -1 & 1 \end{pmatrix}$$

e pertanto avremo la soluzione

$$\begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \frac{1}{x_1 - x_0} \begin{pmatrix} x_1 f(x_0) - x_0 f(x_1) \\ -f(x_0) + f(x_1) \end{pmatrix} .$$

Quindi il polinomio $p_1(x)$ è

$$p_1(x) = \frac{x_1 f(x_0) - x_0 f(x_1)}{x_1 - x_0} + \frac{f(x_1) - f(x_0)}{x_1 - x_0} x$$

ed evidenziando $f(x_0)$, $f(x_1)$ possiamo anche riscriverlo in forma di Lagrange

$$p_1(x) = f(x_0)\frac{x_1 - x}{x_1 - x_0} + f(x_1)\frac{x - x_0}{x_1 - x_0}$$

dove sono ben visibili i polinomi $l_0$, $l_1$.

Le funzioni Matlab/Octave `lagrai.m` e `lagrai_target.m` di Appendice C, consentono di calcolare l'$i$-esimo polinomio elementare di Lagrange nel punto $x$ o su un vettore di valori, rispettivamente.

Nel caso di nodi $x_i$ equispaziati, cioè $x_i - x_{i-1} = h$ ovvero $x_i = x_0 + i\,h, \quad i = 0, ..., n$, i polinomi elementari assumono una forma particolarmente interessante dal punto di vista implementativo.

Con il cambio di variabile $x(t) = x_0 + t\,h,\ t = 0, \ldots, n$, $l_i(x)$ sarà una funzione di $t$, ovvero

$$l_i(t) = \prod_{j=0, j\neq i}^{n} \frac{x_0 + t\,h - x_0 - j\,h}{x_0 + i\,h - x_0 - j\,h} = \prod_{j=0, j\neq i}^{n} \frac{t-j}{i-j}\,.$$

Detta ora $\omega_{n+1}(t) = t(t-1)\cdots(t-n)$, risulta

$$\prod_{j=0, j\neq i}^{n} (t-j) = \frac{\omega_{n+1}(t)}{t-i}\,, \tag{5.6}$$

$$\prod_{j=0, j\neq i}^{n} (i-j) = \prod_{j=0}^{i-1}(i-j) \cdot \prod_{j=i+1}^{n} (i-j) = (-1)^{n-i} i!(n-i)!\,, \tag{5.7}$$

da cui

$$p_n(t) = \frac{\omega_{n+1}(t)}{n!} \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} \frac{y_i}{(t-i)}\,. \tag{5.8}$$

Quando i nodi sono equispaziati, è facile verificare che per il calcolo di $p_n(t)$ sono necessarie $n^2/2$ addizioni e $4n$ moltiplicazioni.

Esercizio 46. *Costruire la funzione Matlab/Octave che calcola l'i-esimo polinomio elementare di Lagrange su nodi equispaziati facendo uso delle formule (5.6) e (5.7). Costruire quindi il polinomio interpolante mediante la (5.8).*

### 5.2.1 Analisi dell'errore d'interpolazione

Sia $f(x)$ definita su $[a, b]$. Detto $p_n(x)$ il polinomio d'interpolazione sugli $n+1$ punti a due a due distinti $x_0, ..., x_n$, indicheremo con

$$r_{n+1}(x) = f(x) - p_n(x)$$

la *funzione errore* per la quale si ha $r_{n+1}(x_i) = 0,\ i = 0, ..., n$.

**Teorema 16.** *Se $f \in \mathcal{C}^{n+1}[a, b]$, allora*

$$r_{n+1}(x) = (x - x_0)\cdots(x - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!}\,, \tag{5.9}$$

*con $\xi \in (a, b)$ e $x_i$ distinti.*

**Dim.** Se $x = x_i$, l'errore è nullo e quindi il risultato è ovvio.

Sia ora $x \neq x_i$, allora preso un qualsiasi $t \in I_x$ ($I_x$ il più piccolo intervallo che contiene i punti $x_0, ..., x_n, x$) possiamo definire la funzione *ausialiaria*

$$g(t) = r_{n+1}(t) - \frac{\omega_{n+1}(t) r_{n+1}(x)}{\omega_{n+1}(x)} .$$

Poiché $f \in \mathcal{C}^{n+1}(I_x)$ segue che $g \in \mathcal{C}^{n+1}(I_x)$ ed ha $n+2$ zeri distinti in $I_x$. Infatti $g(x_i) = 0,\ i = 0, \ldots, n$ e $g(x) = 0$. Allora per il teorema di Rolle, $g'$ ha $n+1$ zeri e così via finché $g^{(n+1)}$ ha un solo zero, che indichiamo con $\xi$.

Ma $r_{n+1}^{(n+1)}(t) = f^{(n+1)}(t)$ e $\omega_{n+1}^{(n+1)} = (n+1)!$ pertanto

$$g^{(n+1)}(t) = f^{(n+1)}(t) - \frac{(n+1)!}{\omega_{n+1}(x)} r_{n+1}(x) .$$

Quando valutiamo questa espressione in $t = \xi$ otteniamo il risultato richiesto. □

Esempio 32. Calcoliamo l'*errore nell'interpolazione lineare*. Dal Teorema 5.9, $r_2(x) = (x - x_0)(x - x_1)\frac{f''(\xi)}{2},\ \ \xi \in (x_0, x_1)$. Ora,

$$\max_{x \in (x_0, x_1)} |(x - x_0)(x - x_1)| = \frac{(x_0 - x_1)^2}{4} .$$

Se indichiamo con $M_2 = \max_{x \in (x_0, x_1)} |f''(x)|$, possiamo dare la seguente maggiorazione

$$|r_2(x)| \leq M_2 \frac{(x_0 - x_1)^2}{8} .$$

Ad *esempio*, per $f(x) = \tan(x),\ x \in [1.35, 1.36]$, sapendo che $f''(x) = \dfrac{2 \sin x}{\cos^3 x}$, troveremo $|r_2(x)| \leq 0.24 \cdot 10^{-2}$.

Esempio 33. Per approssimare $\sqrt{\bar{x}}$, dove $\bar{x}$ non è un quadrato perfetto, possiamo interpolare come segue. Siano $x_0, x_1, x_2$ i quadrati perfetti più vicini ad $\bar{x}$, dei quali consideriamo le loro radici. Si tratta quindi di costruire il polinomio di grado 2, $p_2(x)$, interpolante i dati $(x_i, \sqrt{x_i}),\ i = 0, 1, 2$. Allora $\sqrt{\bar{x}} \approx p_2(\bar{x})$.

Se $\bar{x} = 0.6$, consideriamo i tre punti $(0.49, \sqrt{0.49} = 0.7),\ (0.64, \sqrt{0.64} = 0.8),\ (0.81, \sqrt{0.81} = 0.9)$ ed il relativo polinomio d'interpolazione $p_2(x)$. È facile provare che $p_2(0.6) \approx 0.774$ è un'approssimazione di $\sqrt{0.6}$ (vedi Figura 5.3).

Circa l'errore che commettiamo, si tratta di stimare l'errore in [0.49, 0.81]. Ora essendo $\omega_3(x) = (x - 0.49)(x - 0.64)(x - 0.81)$, $g(x) = \sqrt{x}$ t.c. $g^{(3)}(x) = \frac{3}{8\sqrt{x^5}}$, l'errore in modulo è

$$|r_3(x)| \leq |\omega_3(x)| \frac{3}{3!\, 8\sqrt{\xi^5}},\ \ \xi \in [0.49, 0.81] .$$

Infine,

$$|r_3(0.6)| \leq \frac{0.924 \cdot 10^{-3}}{16\sqrt{(0.49)^5}} \leq 0.35 \cdot 10^{-3}\,.$$

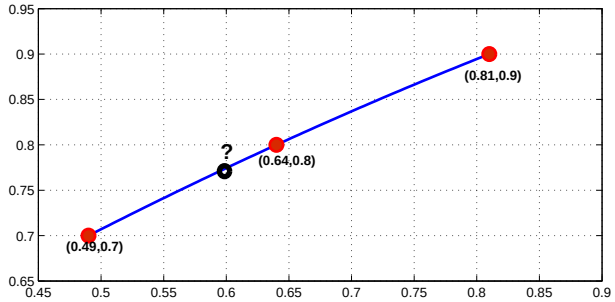


**Figura** 5.3: La parabola dell'Esempio 33.

## 5.3 Errore d'interpolazione e fenomeno di Runge

Se $x_i = x_{i-1} + h,\ \ i = 1, \ldots, n$, con $x_0$ dato e $h > 0$, si può provare che

$$\left| \prod_{i=0}^{n} (x - x_i) \right| \leq n!\, \frac{h^{n+1}}{4}\,.$$

La dimostrazione, per induzione, procede come segue.

- Per $n = 1$, abbiamo due punti $x_0, x_1$ distanti $h$. La funzione $|(x - x_0)(x - x_1)|$, come visto nell'Esempio 32, è una parabola il cui massimo, nel vertice, vale $h^2/4$. Questo prova il passo iniziale dell'induzione.
- Sia vera per $n + 1$ punti, $x_0, \ldots, x_n$. Prendiamo un altro punto $x_{n+1}$. Avremo

$$|\prod_{i=0}^{n+1}(x - x_i)| = |\prod_{i=0}^{n}(x - x_i)| \cdot |x - x_{n+1}| \underbrace{\leq}_{ind.} n!\frac{h^{n+1}}{4}(n+1)h = (n+1)!\frac{h^{n+2}}{4}\,.$$

Pertanto l'errore d'interpolazione, per punti equispaziati, si maggiora come segue:

$$\max_{x \in I} |r_{n+1}(x)| \leq \max_{x \in I} \left| f^{(n+1)}(x) \right| \frac{h^{n+1}}{4(n+1)}\,. \tag{5.10}$$

La disuguaglianza (5.10) ci dice che nonostante per $h \to 0$, $\lim_{h\to 0} \frac{h^{n+1}}{4(n+1)} = 0$ l'errore non è detto vada a zero. Ciò dipende dall'ordine di infinito della derivata $n+1$-esima rispetto al fattore $\frac{h^{n+1}}{4(n+1)}$. Vediamolo su un esempio.

Riconsideriamo l'Esempio 29, ovvero la *funzione di Runge*:

$$g(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5] \tag{5.11}$$

Sugli $n+1$ punti equispaziati $x_i = -5+i\,h,\ i = 0, ..., n,\ \ h = 10/n$ consideriamo il polinomio d'interpolazione di grado $n$. Runge osservò che nonostante $g \in \mathcal{C}^\infty(\mathbb{R})$, l'errore $g(x) - p_n(x)$ tende a crescere con $n$. In particolare dimostrò che se $|x| > 3.63$ il polinomio si discosta sempre più dalla funzione oscillando enormemente. In Figura 5.4 si vede la funzione di Runge (scalata in [-1,1]) e il suo polinomio d'interpolazione di grado 10 su nodi equispaziati e nodi di *Chebyshev* (che descriveremo oltre).

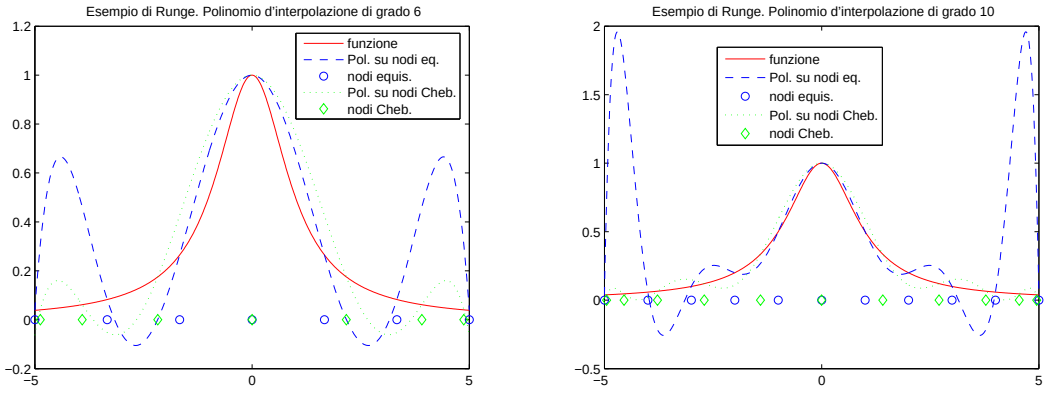


**Figura** 5.4: Funzione di Runge e polinomio d'interpolazione su nodi equispaziati e di Chebyshev.

- **Prima domanda**: come evitare il fenomeno di Runge?

  Una prima risposta è quella di usare i *punti di Chebyshev* invece dei punti equispaziati. In $[-1, 1]$ i nodi di Chebyshev sono gli *zeri* del polinomio ortogonale di Chebsyhev (di prima specie) di grado $n$

$$x_k^{(C)} = \cos\left(\frac{2k-1}{n}\frac{\pi}{2}\right), \quad k = 1, \ldots, n\,. \tag{5.12}$$

  Osserviamo che, se invece di $[-1, 1]$ consideriamo il generico intervallo $[a, b]$, allora applicando la trasformazione lineare che manda l'intervallo $[-1, 1]$ in $[a, b]$ i punti corrispondenti sono

$$x_k = \frac{a+b}{2} + \frac{b-a}{2} x_k^{(C)}$$

dove $x_k^{(C)}$ sono i nodi di Chebsyehv in $[-1,1]$. In alcuni testi (vedasi ad es. [19, p. 78]) si considerano come nodi di Chebyshev i punti di `Chebyshev-Lobatto`, $x_k^{(CL)} = \cos(k\pi/n),\ k = 0,\ldots,n$, che includono pure gli estremi dell'intervallo.

Da un punto di vista geometrico, i punti di Chebyshev sono le proiezioni sull'intervallo $[-1,1]$ dei punti equispaziati sul semicerchio di diametro $[-1,1]$ (vedasi Figura 5.5). Come si vede dal grafico, se prendiamo 2 punti $a$ e $b$ di $[-1,1]$ ottenuti come proiezione di punti equispaziati sul semicerchio, vale la relazione

$$|\arccos(a) - \arccos(b)| = |\theta_a - \theta_b| = \text{cost}\,.$$

Questo dice che *i punti di Chebyshev hanno la distribuzione dell'arcocoseno.*

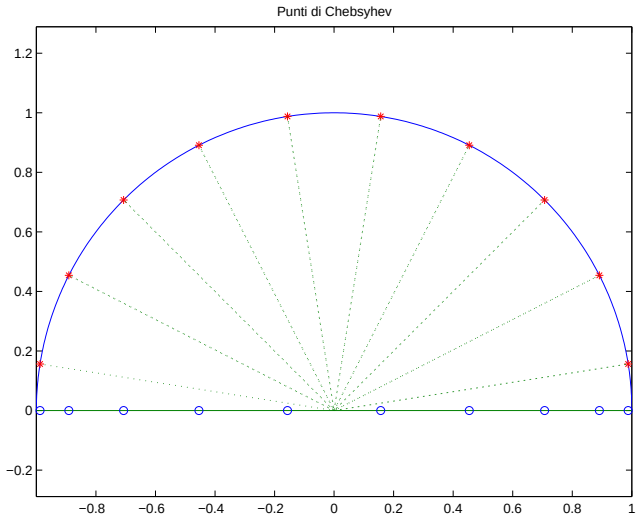


**Figura** 5.5: 10 punti di Chebyshev.

- **Seconda domanda**: perchè i punti di Chebyshev sono migliori di quelli equispaziati?

  Una prima risposta è grafica (vedasi Figura 5.4)... ma non basta! Nella prossima sessione presenteremo una risposta matematicamente più formale basata sullo studio della `costante di Lebesgue`.

### 5.3.1 La costante di Lebesgue

Indichiamo con $X$ la matrice triangolare inferiore di dimensione infinita, i cui elementi $x_{i,j}$ sono punti appartenenti all'intervallo $[a,b]$. Inoltre per ogni $n \geq 0$, la riga $n$-esima ha $n+1$ elementi (corrispondenti ai punti su cui costruiremo il polinomio d'interpolazione di grado $n$). Sia $p_n^f(x)$ il polinomio di grado $n$ che interpola la funzione $f$ usando gli $n+1$ nodi della $n$-esima riga di $X$.

Fissata $X$ e la funzione $f$, indichiamo con

$$E_{n,\infty}(X) = \|f - p_n^f\|_\infty, \;\; n = 0, 1, .... \tag{5.13}$$

l'errore in norma infinito tra $f$ e il suo polinomio d'interpolazione. Indichiamo con $p_n^* \in \mathbb{P}_n$ il `polinomio di migliore approssimazione uniforme` di $f$ per il quale consideriamo

$$E_n^* = \|f - p_n^*\|_\infty \leq \|f - q_n\|_\infty, \;\; \forall\, q_n \in \mathbb{P}_n\,. \tag{5.14}$$

**Teorema 17.** *Sia $f \in \mathcal{C}[a,b]$ e $X$ come prima. Allora*

$$E_{n,\infty}(X) \leq (1 + \Lambda_n(X))E_n^*, \;\; n = 0, 1, \ldots \tag{5.15}$$

*con*

$$\Lambda_n(X) = \max_{x\in[a,b]} \sum_{i=1}^{n} |l_i(x)|$$

*che si chiama* `costante di Lebesgue`.

**Osservazione**. È facile provare che $\Lambda_n(X) \geq 1$.

Ora, dal Teorema 17, $E_n^*$ è indipendente da $X$ mentre non lo è $E_{n,\infty}(X)$. Pertanto la scelta del vettore dei nodi è *fondamentale* per il valore che può assumere $\Lambda_n$. Si dimostra (vedi ad esempio [21]) che la crescita della costante di Lebesgue per nodi equispaziati $X_e$ e nodi di Chebyshev $X_c$ è come segue:

$$\begin{aligned} \Lambda_n(X_e) &\approx \frac{2^{n-1}}{n\,\mathrm{e}\,\log_e n} \\ \Lambda_n(X_c) &\approx \frac{2}{\pi}\,\log_e n\,. \end{aligned}$$

In entrambi i casi per $n \to \infty$ la costante di Lebesgue tende ad infinito, ma per i nodi di Chebyshev la crescita è logaritmica invece che esponenziale. Inoltre, è stato dimostrato che, per ogni $n \geq 1$, $\Lambda_n(X_c) \leq \sigma(n)$ con $\sigma(n) = \frac{2}{\pi}\log(n+1) + 1$ .

Per avere un'idea dell'andamento della costante di Lebesgue per punti di Chebyshev, in Tabella 5.1 sono riportati alcuni valori di $\Lambda_n(X_c)$ e di $\sigma(n)$ (arrotondati alla terza cifra decimale). I valori di $\Lambda_n(X_c)$ sono stati calcolati con la funzione `CostLebesgue.m` (vedi appendice C).

Come ultima osservazione, *i nodi di Chebyshev sono nodi quasi-ottimali*. Infatti, sempre in [21, pag. 101], si osserva che esiste un'insieme $X^*$ di nodi ottimali anche se il loro calcolo non è semplice. Ma, come provato da L. Brutman in SIAM J. Numer. Anal. 15 (1978), l'insieme dei *nodi di Chebsyhev espansi*, che altro non sono che i nodi di *Chebyshev-Lobatto* $x_i^{(C_e)} = \cos\left(\dfrac{\pi\, i}{n}\right)$ $i = 0, \ldots, n$ è utile per tutti gli usi più frequenti e quindi essi possono essere considerati come nodi ottimali sull'intervallo.

| n | $\Lambda_n(X_c)$ | $\sigma(n)$ |
|---|---|---|
| 2 | 1.189 | 1.699 |
| 5 | 1.828 | 2.141 |
| 10 | 2.232 | 2.527 |
| 15 | 2.412 | 2.765 |
| 20 | 2.477 | 2.938 |

Tabella 5.1: Confronti dei valori della costante di Lebesgue per punti di Chebsyshev e della funzione $\sigma(n)$

### 5.3.2 Stabilità dell'interpolazione polinomiale

Dati $(x_i, f(x_i)),\ i = 0, \ldots, n$, invece dei valori $f(x_i)$ consideriamo dei valori perturbati $\tilde{f}(x_i)$. Indichiamo con $p_n^{\tilde{f}}$ il polinomio di grado $n$ che interpola $(x_i, \tilde{f}(x_i))$, $i = 0, \ldots, n$. Allora,

$$\|p_n^f - p_n^{\tilde{f}}\|_\infty = \max_{x\in[a,b]} \left| \sum_{j=0}^{n} l_j(x) \left( f(x_j) - \tilde{f}(x_j) \right) \right| \tag{5.16}$$

$$\leq \max_{x\in[a,b]} \sum_{j=0}^{n} |l_j(x)| \max_{0\leq i\leq n} |f(x_i) - \tilde{f}(x_i)| \,. \tag{5.17}$$

Essendo $\Lambda_n(X) := \max_{x\in[a,b]} \sum_{j=0}^{n} |l_j(x)|$, con $X = \{x_0, \ldots, x_n\}$, la costante di Lebesgue si può interpretare come il *numero di condizionamento* del problema d'interpolazione polinomiale.

Esempio 34. Consideriamo la funzione $f(x) = \sin(2\pi x),\ x \in [-1, 1]$ che desideriamo interpolare su 22 nodi equispaziati. Consideriamo poi i valori perturbati $\tilde{f}(x_i)$ tali che

$$\delta := \max_{0\leq i\leq 21} |f(x_i) - \tilde{f}(x_i)| \approx 4.5\, 10^{-2} \,.$$

Costruiamo $p_{21}^f$ e $p_{21}^{\tilde{f}}$. Mediante la (5.17) otteniamo la stima $\Lambda_{21} \approx 4500$. Ovvero il problema è sensibile alle perturbazioni (che ritroviamo soprattutto nei punti di massima oscillazione della funzione).

## 5.4 Polinomio interpolante in forma di Newton

Si può esprimere il polinomio di interpolazione di grado $n$ della funzione $f$, $p_n^f(x)$, in *forma di Newton*:

$$p_n^f(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)\cdots(x - x_{n-1}) \,,$$

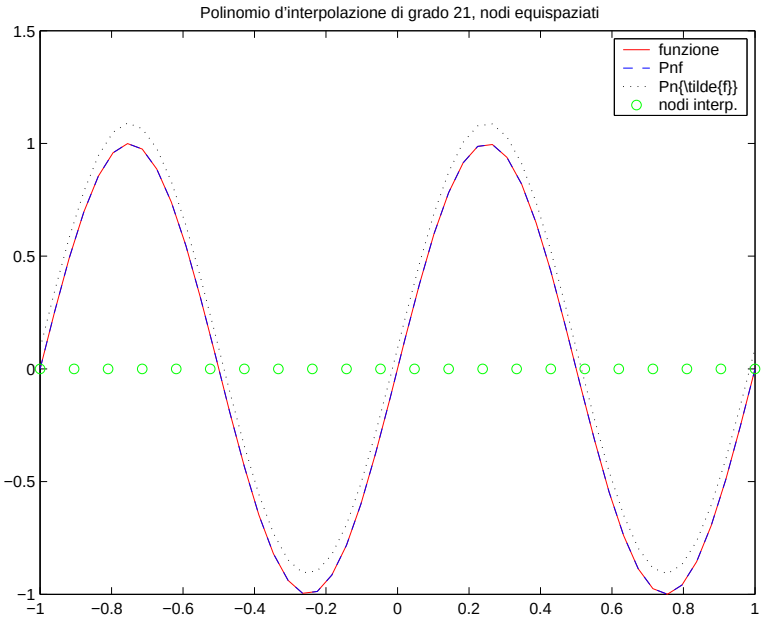


**Figura** 5.6: Funzione dell'Esempio 34

dove $b_i$ rappresenta la *differenza divisa* di ordine $i$ della funzione $f$. Per comprendere come costruire sifatto polinomio, dobbiamo dapprima introdurre le differenze divise di una funzione.

### 5.4.1 Differenze divise e loro proprietà

**Definizione 20.** *Dati $n+1$ punti distinti $x_0, \ldots, x_n$ e i valori $y_i = f(x_i)$, la differenza divisa di ordine 0 della funzione $f$ è $f[x_i] = f(x_i)$, la differenza divisa di ordine 1 è $f[x_i, x_{i+1}] = \dfrac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$ e ricorsivamente, la differenza di ordine $k$ è*

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \,. \tag{5.18}$$

1. La differenza divisa di ordine $k+1$ di un polinomio di grado $k$ è identicamente nulla. Vediamolo su un esempio.

   Esempio 35. Consideriamo i punti $\{-1, 2, 3, 5\}$ e $f(x) = x^2 + 1$. La tabella delle differenze divise è la seguente

| $x_i$ | $y_i$ | ordine 1 | ordine 2 | ordine 3 |
|---|---|---|---|---|
| -1 | 2 | | | |
| 2 | 5 | 1 | | |
| 3 | 10 | 5 | 1 | |
| 5 | 26 | 8 | 1 | 0 |

Tabella 5.2: Differenze divise della funzione $x^2 + 1$

2. La differenza divisa di ordine $n$, si esprime come

$$f[x_0, \ldots, x_n] = \sum_{j=0}^{n} (-1)^j \frac{f(x_j)}{\prod_{i=0, i>j}(x_i - x_j)} \,. \tag{5.19}$$

   La verifica viene lasciata come esercizio.

3. Le differenze divise sono invariati rispetto all'ordine dei nodi. Ovvero

$$f[x_0, \ldots, x_k] = f[x_{i_0}, \ldots, x_{i_k}] \tag{5.20}$$

   dove $(i_0, \ldots, i_k)$ è una qualsiasi permutazione di $(0, \ldots, k)$. Questa proprietà è una diretta conseguenza della formula (5.19).

Di tutte le predette proprietà, il lettore curioso può trovare la dimostrazione ad esempio in [2, pag. 384 e ss.].

Possiamo osservare che il polinomio

$$p_n^f(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}) \tag{5.21}$$

è interpolante la funzione $f(x)$ nei punti $x_i$: infatti $p_n(x_i) = f(x_i)$. Per l'unicità del polinomio d'interpolazione possiamo dire che esso *è il polinomio d'interpolazione!*

La (5.21) è detta la *forma di Newton* del polinomio d'interpolazione che potremo riscrivere più semplicemente come

$$p_n^f(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0) \cdots (x - x_{n-1}) \,, \tag{5.22}$$

dove $b_i$ rappresenta la *differenza divisa* di ordine $i$ della funzione $f$.

**Algoritmo delle differenze divise**

Per determinare i coefficienti $b_i,\ i = 0, ..., n$ in (5.22), l'*algoritmo delle differenze divise* è descritto nella funzione `DiffDivise.m` (cfr. Appendice C).

Alla fine, il valore del polinomio $p_n$ in $x$, si determina con lo *schema di Hörner*:

$$\begin{aligned} p &= b_n; \\ p &= (x - x_i)p + b_i, \;\; i = n-1, ..., 0 \end{aligned} \tag{5.23}$$

L'errore d' interpolazione si può esprimere usando le differenze divise di ordine $n+1$ della funzione $f$. Ricordando che per l'errore vale

$$E_n(f) = f(x) - p_n^f(x) = \left( \prod_{i=0}^{n} (x - x_i) \right) \frac{f^{(n+1)}(\xi)}{(n+1)!} \,. \tag{5.24}$$

facciamo ora vedere il legame esistente con le differenze divise. Vale il seguente risultato:

**Proposizione 10.** *Se $f \in \mathcal{C}^{n+1}(I)$, allora*

$$\frac{f^{(n+1)}(\xi)}{(n+1)!} = f[x_0, ..., x_n, x] \,, \tag{5.25}$$

*con $\xi$ punto che appartiene al più piccolo intervallo che contiene $x_0, \ldots, x_n, x$.*

**Dim.** Dati $x_0, \ldots, x_n$ e i valori corrispondenti della funzione $f(x_i),\ i = 0, ..., n$, indichiamo con $p_n^f$ il polinomio d'interpolazione. Preso poi un punto $x$, diverso dai punti $x_i$, che potremo considerare come un altro punto $x_{n+1}$ d'interpolazione, costruiamo $p_{n+1}^f$. Grazie alla formula di Newton dell'interpolante

$$p_{n+1}^f(t) = p_n^f(t) + (t - x_0) \cdots (t - x_n) f[x_0, \ldots, x_n, t] \,.$$

Per $t = x$, $p_{n+1}^f(x) = f(x)$, da cui

$$\begin{aligned} E_n(x) &= f(x) - p_n^f(x) = p_{n+1}^f(x) - p_n^f(x) \\ &= (x - x_0) \cdots (x - x_n) f[x_0, \ldots, x_n, x] \\ &= \omega_{n+1}(x) f[x_0, \ldots, x_n, x] \,. \end{aligned} \tag{5.26}$$

Essendo $f \in \mathcal{C}^{n+1}(I)$ e ricordando che

$$E_n(x) = (x - x_0) \cdots (x - x_n) \frac{f^{(n+1)}(\xi)}{(n+1)!} \,, \tag{5.27}$$

dal confronto di (5.26) e (5.27) si conclude. □

◇◇

### 5.4.2 Formula di Hermite-Genocchi per le differenze divise

Questa formula è interessante perchè permette di estendere la forma di Newton dell'interpolante anche al caso di punti ripetuti.

**Teorema 18.** *Siano dati $n+1$ punti distinti $x_0, x_1, ..., x_n$ e sia $f \in \mathcal{C}^n(I)$ con $I$ il più piccolo intervallo che li contiene. Allora*

$$f[x_0, x_1..., x_n] = \int_{\tau_n} f^{(n)}(x_0 t_0 + \cdots + x_n t_n) dt_1 dt_2 \dots dt_n \ , \tag{5.28}$$

*dove $t_0 = 1 - (t_1 + t_2 + ... + t_n)$ e l'integrale è fatto sul simplesso*

$$\tau_n = \left\{ (t_1, t_2, \dots, t_n) \ : \ t_i \geq 0, \sum_{i=1}^{n} t_i \leq 1 \right\}.$$

**Dim.** Come prima cosa, osserviamo che $t_0 \geq 0$ e $\sum_{i=0}^{n} t_i = 1$.

La dimostrazione di (5.28) si fa per induzione su $n$.

1. Se $n = 1$, $\tau_1 = [0, 1]$. L'equazione (5.28) diventa

$$\begin{aligned} \int_0^1 f'(t_0 x_0 + t_1 x_1) dt_1 &= \int_0^1 f'(x_0 + t_1(x_1 - x_0)) dt_1 \\ &= \frac{1}{x_1 - x_0} f(x_0 + t_1(x_1 - x_0))|_{t_1=0}^{t_1=1} \\ &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1] \ . \end{aligned}$$

2. Nel caso $n = 2$, $\tau_2$ è il triangolo di vertici $(0,0), (1,0), (0,1)$.

$$\begin{aligned} \int_{\tau_2} f''(t_0 x_0 &+ t_1 x_1 + t_2 x_2) dt_1 dt_2 = \\ &= \int_0^1 \int_0^{1-t_1} f''(x_0 + t_1(x_1 - x_0) + t_2(x_2 - x_0)) dt_2 dt_1 \\ &= \int_0^1 \frac{1}{x_2 - x_0} f'(x_0 + t_1(x_1 - x_0) + t_2(x_2 - x_0))|_{t_2=0}^{t_2=1-t_1} dt_1 \\ &= \frac{1}{x_2 - x_0} \left\{ \int_0^1 f'(x_2 + t_1(x_1 - x_2)) dt_1 - \right. \\ &- \left. \int_0^1 f'(x_0 + t_1(x_1 - x_0)) dt_1 \right\} \\ &= \frac{1}{x_2 - x_0} \{ f[x_1, x_2] - f[x_0, x_1] \} = f[x_0, x_1, x_2] \ . \end{aligned}$$

3. Nel caso generale si integrerà prima rispetto ad una variabile per ridurre la dimensione, quindi per ipotesi induttiva si conclude.

Questo conclude la dimostrazione. □

Nel caso in cui tutti punti "collassano" in $x_0$ (ciò ha senso poichè la funzione $f[x_0, ..., x_n]$ è continua nelle sue variabili) usando le proprietà delle differenze divise avremo

$$f[\underbrace{x_0, ..., x_0}_{n+1\ volte}] = \int_{\tau_n} f^{(n)}(x_0) dt_1 \cdots dt_n = f^{(n)}(x_0) \cdot \mathrm{Vol}_n(\tau_n)$$

ove $\mathrm{Vol}_n(\tau_n)$ indica il volume $n$-dimensionale del simplesso $\tau_n$. Ora, ricordando la relazione $\mathrm{Vol}_n(\tau_n) = \dfrac{1}{n!}$ si ottiene la formula

$$f[\underbrace{x_0, \ldots, x_0}_{n+1\ volte}] = \frac{f^{(n)}(x_0)}{n!}\,. \tag{5.29}$$

ESEMPIO 36. Se di una funzione $f(x)$ si conoscono il valore in un punto $x_0$ e i valori delle derivate fino all'ordine $k$ in $x_0$, la tabella delle differenze divise è la Tabella 5.3. dove

| | | | | | | |
|---|---|---|---|---|---|---|
| $x_0$ | $f[x_0]$ | $f[x_0, x_0]$ | $\rightarrow$ | $f[x_0, x_0, x_0]$ | $\ldots$ | $f[\underbrace{x_0, \ldots, x_0}_{k+1\ volte}]$ |
| $x_0$ | $f[x_0]$ | $f[x_0, x_0]$ | $\nearrow$ | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | | | | |
| $\vdots$ | $\vdots$ | $\vdots$ | | $f[x_0, x_0, x_0]$ | | |
| $x_0$ | $f[x_0]$ | $f[x_0, x_0]$ | | | | |

Tabella 5.3: Tabella delle differenze divise per un punto ripetuto $k+1$ volte

$f[x_0, x_0] = f'(x_0)$, $f[x_0, x_0, x_0] = \dfrac{f''(x_0)}{2}$ e $f[\underbrace{x_0, \ldots, x_0}_{k+1\ volte}] = \dfrac{f^{(k)}(x_0)}{k!}$. In questo caso quindi il polinomio d'interpolazione in forma Newton coinciderà con la formula di Taylor.

### 5.4.3 Interpolazione di Hermite

Il *polinomio osculatore di Hermite* (dal latino, *osculare* che significa *baciare*) è quel polinomio $p_{2n+1}(x)$, di grado $2n+1$ costruito usando $n+1$ distinti $x_i$, $i = 0, ..., n$ che soddisfa le $2n+2$ condizioni

$$p_{2n+1}(x_i) = f(x_i),\ \ p'_{2n+1}(x_i) = f'(x_i)\,,\ i = 0, \ldots, n\,. \tag{5.30}$$

In forma di Lagrange, il polinomio osculatore di Hermite si scrive come segue:

$$p_{2n+1}(x) = \sum_{i=0}^{n} u_i(x) f(x_i) + \sum_{i=0}^{n} v_i(x) f'(x_i) \tag{5.31}$$

dove i polinomi $u_i(x)$ e $v_i(x)$ sono funzioni dei polinomi elementari di Lagrange, ovvero

$$u_i(x) = [1 - l_i'(x_i)(x - x_i)] l_i^2(x), \tag{5.32}$$

$$v_i(x) = (x - x_i) l_i^2(x)\,. \tag{5.33}$$

È facile verificare che i polinomi $u_i(x)$ e $v_i(x)$ hanno grado $2n+1$ e per essi valgono le condizioni

$$\begin{gathered} u_i(x_k) = \delta_{i,k}, \\ v_i(x_k) = 0,\ u_i'(x_k) = 0,\ \forall\, k \\ v_i'(x_k) = \delta_{i,k}\,. \end{gathered}$$

Ne segue che il polinomio (5.31) ha grado $\leq 2n+1$ e soddisfa le condizioni d'interpolazione (5.30).

Il polimomio (5.31) si può costruire anche in forma di Newton mediante la seguente tabella delle differenze divise: dove $f[x_i, x_i] = f'(x_i),\ i = 0, ..., n$. Per suggerimenti imple-

| | | | | | |
|---|---|---|---|---|---|
| $x_0$ | $f[x_0]$ | | | | |
| | | $f[x_0, x_0]$ | | | |
| $x_0$ | $f[x_0]$ | | $f[x_0, x_0, x_1]$ | | |
| | | $f[x_0, x_1]$ | | $\ddots$ | |
| $x_1$ | $f[x_1]$ | | $f[x_0, x_1, x_1]$ | | |
| | | $f[x_1, x_1]$ | | | |
| $x_1$ | $f[x_1]$ | $\vdots$ | $\vdots$ | | $f[x_0, x_0, \ldots, x_n, x_n]$ |
| | | | $\vdots$ | | |
| $\vdots$ | $\vdots$ | $\vdots$ | $f[x_0, x_0, x_n]$ | | |
| | | $f[x_{n-1}, x_n]$ | | | |
| $x_n$ | $f[x_n]$ | | $f[x_0, x_n, x_n]$ | | |
| | | $f[x_n, x_n]$ | | | |
| $x_n$ | $f[x_n]$ | | | | |

Tabella 5.4: Tabella delle differenze divise per l'interpolazione di Hermite

mentativi del polinomio osculatore di Hermite in forma di Newton, rimandiamo all'Esercizio 52.

Possiamo anche estendere la formula dell'errore d'interpolazione,(5.9) o (5.24), al caso in cui il polinomio sia costruito su nodi non necessariamente distinti.

**Teorema 19.** *Se $f(x) \in \mathcal{C}^{n+1}[a,b]$, esiste un punto $\xi = \xi(x) \in (a,b)$ tale che*

$$f[x_0, x_1, \ldots, x_n, x] = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,. \tag{5.34}$$

*Se $f(x) \in \mathcal{C}^{n+2}[a,b]$, esiste un punto $\eta = \eta(x) \in (a,b)$ tale che*

$$\frac{d}{dx} f[x_0, x_1, \ldots, x_n, x] = \frac{f^{(n+2)}(\eta)}{(n+2)!}\,. \tag{5.35}$$

*Infine, se i punti $x_0, \ldots, x_n, x$ sono tutti coincidenti allora $\xi = \eta = x$.*

**Dim.** Applicando il teorema della media integrale alla funzione $f[x_0, x_1, \ldots, x_n, x]$ espressa mediante la formula di Hermite-Genocchi, si ha

$$f[x_0, x_1, \ldots, x_n, x] = f^{(n+1)}(\xi) \int_0^1 dt_1 \int_0^{t_1} dt_2 \ldots \int_0^{t_n} dt_n$$

da cui deriva immediatamente la (5.34). La (5.35) si ottiene in maniera analoga osservando che

$$\frac{d}{dx} f[x_0, x_1, \ldots, x_n, x] = f[x_0, x_1, \ldots, x_n, x, x]\,.$$

Questo conclude la dimostrazione. □

### 5.4.4 Algoritmo iterativo di Neville

Dati $a, b$, estremi dell'intervallo di interpolazione, $n$ il numero dei nodi di interpolazione, $\mathbf{x}, \mathbf{y}$ array di dimensione $n$ che contengono i nodi equispaziati e il valore della funzione $\mathbf{f} = (f(x_0), \ldots, f(x_n))$ nei nodi equispaziati $x_i = a + (b-a)\frac{i}{n}$, $i = 0, ..., n$. Posto $y_i = f(x_i)$, indichiamo con $x \in [a,b]$ il punto su cui valutare il polinomio interpolante allora il polinomio interpolante in $x$ è ottenuto con la seguente formula iterativa, nota come *schema d'interpolazione di Neville*:

$$p_i = y_i, \quad i = 0, \ldots, n;$$

allora

$$p_{0,\ldots,k}(x) = \frac{p_{1,\ldots,k}(x)(x - x_0) - p_{0,\ldots,k-1}(x)(x - x_k)}{x_k - x_0} \tag{5.36}$$

è il polinomio d'interpolazione su $x_0, \ldots, x_k$. Pertanto alla fine, $p_{0,1,\ldots,n}(x)$ rappresenterà il valore in $x$ del polinomio d'interpolazione di grado $n$ costruito mediante l'uso dei punti $(x_i, y_i)$.

Il vantaggio di questa tecnica, è che il polinomio d'interpolazione viene costruito come una successione di polinomi di grado crescente, per cui il procedimento si arresterà quando è raggiunto il grado richiesto.

In Tabella 5.5 riportiamo lo schema triangolare di Neville nel caso cubico. Come si può facilmente verificare, i polinomi della prima riga $p_{0,\dots,s},\ s=0,\dots,3$ hanno grado crescente da 0 a 3.

| | | | | |
|---|---|---|---|---|
| $x_0$ | $y_0$ | $p_{0,1}(x)$ | $p_{0,1,2}(x)$ | $p_{0,1,2,3}(x)$ |
| $x_1$ | $y_1$ | $p_{1,2}(x)$ | $p_{1,2,3}(x)$ | |
| $x_2$ | $y_2$ | $p_{2,3}(x)$ | | |
| $x_3$ | $y_3$ | | | |

Tabella 5.5: Schema di Neville, per $n=3$.

ESERCIZIO 47. Si valuti il polinomio di Neville di grado $3 \le n \le 10$, interpolante la funzione

$$f(x) = \frac{\sin(x)}{(1+e^x)}, \quad x \in [0, 2\pi],$$

su punti dell'intervallo e si determini anche l'errore d'interpolazione. Fare il grafico della funzione, dell'interpolante di Neville e quello dell'errore al variare di $n$.

## 5.5 Interpolazione polinomiale a tratti: cenni

L' idea sottostante a questa tecnica è di limitare il grado del polinomio di interpolazione aumentando la flessibilità dell'interpolante.

Si parte da una suddivisione $\Delta = \bigcup_{i=1}^{n} I_i$, dove $I_i$ è il generico sottointervallo in cui si è suddiviso l'intervallo $[a,b]$, e si opera un'approssimazione polinomiale di grado basso su ogni sottointervallo $I_i$. Rispetto all'interpolazione (globale) su tutto l'intervallo, pur perdendo in regolarità, questa tecnica migliora la descrizione della funzione da approssimare.

È assai diffuso l'uso dell' *interpolazione lineare a tratti* che genera una funzione che si presenta come un segmento di retta su ogni sottointervallo e come una spezzata sull'intero intervallo dove risulta continua ma non necessariamente derivabile. Dati i punti $x_0,\dots,x_n$ (non necessarimente equispaziati) di $I=[x_0,x_n]$ e i valori $f(x_i)$, $i=0,\dots,n$, indichiamo con $I_i=[x_i,x_{i+1}]$ l'i-esimo intervallino. Su ognuno degli $n$ sotto-intervalli $I_i$, $i=0,...,n-1$, iniziamo con l'approssimare $f$ con un *polinomio lineare a tratti*. Ovvero, su $I_i$, costruiamo

$$p^f_{1,h_i}(x) = f(x_i) + (x-x_i)\,\frac{f(x_{i+1})-f(x_i)}{x_{i+1}-x_i}\,, \tag{5.37}$$

dove l'indice $h_i$ in $p^f_{1,h_i}$ ci ricorda che gli intervallini non hanno tutti la stessa ampiezza.

Posto $H = \max_{0\le i\le n-1} h_i$, vale il seguente risultato.

**Proposizione 11.** *Se $f \in \mathcal{C}^2(I)$, allora*

$$\max_{x\in I} |f(x) - p^f_{1,H}(x)| \leq \frac{H^2}{8} \max_{x\in I} |f''(x)| \,. \tag{5.38}$$

La proposizione dice che se $f$ è sufficientemente regolare allora il polinomio lineare e continuo a tratti converge alla funzione con $H \to 0$.

Facciamo notare come le funzioni `fplot` e `plot` di Matlab costruiscano proprio l'interpolante lineare a tratti.

ESEMPIO 37. Consideriamo i valori della funzione `sin` nei punti equispaziati $x_i = i,\ i = 0, ..., 10$. Usando le seguenti istruzioni Matlab/Octave, facciamo vedere come `plot` produca l'interpolante lineare a tratti (vedasi Figura 5.7).

```
x=1:10; y=sin(x);
xx=0:0.01:10; yy=sin(xx);
plot(x,y,'-',xx,yy,':r')
```

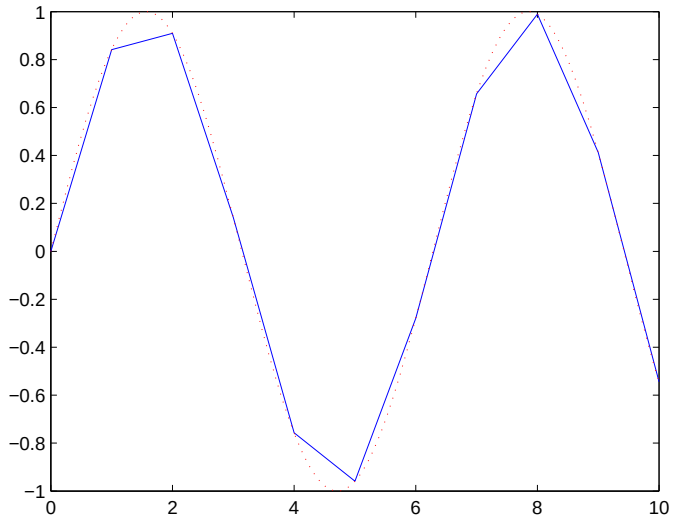


**Figura** 5.7: Funzione seno (linea punteggiata) e la sua interpolante lineare a tratti (linea continua)

La generalizzazione dell'interpolante lineare a tratti è l'interpolazione polinomiale (continua) a tratti.

**Definizione 21.** *$s$ è un polinomio continuo a tratti in [a,b] di grado $k$ se $s \in \mathcal{C}[a,b]$ e se esistono dei punti $\xi_i$, $i = 0, ..., n$ $a = \xi_0 < \xi_1 < \cdots < \xi_n = b$ cosicché $s$ è un polinomio di grado $\leq k$ su ciascun intervallino $[\xi_i, \xi_{i+1}], i = 0, ..., n-1$ .*

Nella sezione che segue introdurremo brevemente le *funzioni splines polinomiali* che rappresentano tuttora uno degli strumenti più flessibili, sia in termini di ordine di approssimazione che di efficienza computazionale, per l'approssimazione sia di funzioni che di dati. In pratica le funzioni splines sono un ottimo compromesso per chi desidera un strumento matematico sufficientemente duttile, efficiente e preciso per approssimare e/o interpolare.

## 5.6 Esercizi proposti

ESERCIZIO 48. `(Appello del 26/9/05)`. *Si consideri la funzione*

$$f(x) = \log(2 + x) \,, \quad x \in [-1, 1] \,.$$

*Indichiamo con $p_n$ il polinomio di interpolazione di grado $\leq n$ costruito usando i punti*

$$x_k = \cos\left(\frac{2k+1}{2n}\,\pi\right), \; k = 0, 1, ..., n$$

*noti come punti di Chebysehv. Sotto tale ipotesi, è noto che l'errore di interpolazione si può maggiorare come segue:*

$$\|f - p_n\|_\infty \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!} 2^{-n} \,. \tag{5.39}$$

1. *Nel caso $n = 4$, si determini una maggiorazione dell'errore usando la (5.39).*
2. *Nel caso in cui il polinomio di interpolazione, sia invece scrivibile in forma in Taylor come*

$$t_n(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f''(0)}{2!}x^2 + \cdots + \frac{f^{(n)}(0)}{n!}x^n \,, \tag{5.40}$$

*l'errore nel generico punto $x$ si esprime come*

$$f(x) - t_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}x^{n+1} \,, \quad -1 < \xi < 1 \,.$$

*Determinare una maggiorazione di*

$$\|f - t_4\|_\infty = \max_{-1 \leq x \leq 1} |f(x) - t_4(x)| \,,$$

*e confrontare il risultato con quello ottenuto nel caso dei punti di Chebyshev.*

3. *Facoltativo: Plottare in un unico grafico, $f(x)$, $p_4(x)$ e $t_4(x)$.*

ESERCIZIO 49. `(Appello del 23/3/06)`. *Si consideri la funzione* $f(x) = x + e^x + \dfrac{20}{1+x^2} - 5$ *ristretta all'intervallo* $[-2, 2]$.

1. *Determinare il polinomio d'interpolazione di grado* 5 *in forma di Newton sui nodi equispaziati* $x_k = -2 + kh,\ k = 0, ..., 5$.
2. *Calcolare l'errore d'interpolazione in un generico punto* $x \in (-2, 2)$.
3. *Ripetere i calcoli usando i punti di Chebyshev.*

ESERCIZIO 50. *Si consideri la funzione* $f(x) = \dfrac{20}{1+\log(x^2)} - 5\,\sin(e^x)$ *ristretta all'intervallo* $[1, 2]$. *Determinare l'unico polinomio (d'interpolazione) di secondo grado,* $p_2(x) = a_0 + a_1 x + a_2 x^2$ *tale che*

$$p_2(0) = f(0),\ p_2(1) = f(1),\ \int_2^2 p_2(x)dx = \int_1^2 f(x)dx\ .$$

*Per il calcolo dell'integrale della funzione usare la funzione* `quadl` *di Matlab, con tolleranza di* $1.e-6$. *Fare quindi il grafico della funzione, del polinomio e di* $\|f - p_2\|_\infty$.

ESERCIZIO 51. `(Appello del 20/7/07)`. *Si consideri la funzione* $f(x) = \cos(x^3)\,(x - 2\,\pi)\,e^{-x},\ x \in [0, \pi]$. *Sperimentalmente si determini il grado del polinomio d'interpolazione, costruito sui nodi di Chebsyhev in* $[0, \pi]$, *che approssima* $f(x)$ *in norma infinito a meno di* $tol = 1.e-4$.

ESERCIZIO 52. *È noto che se* $f \in \mathcal{C}^1[a, b]$ *e* $x_0, ..., x_n$ *sono* $n+1$ *punti distinti esiste un unico polinomio di grado* $2n+1$ *che interpola* $f(x)$ *e* $f'(x)$ *nei punti* $x_i$. *Tale polinomio è quello di Hermite*

$$H_{2n+1}(x) = \sum_{i=0}^{n} \left[ f(x_i) H_{n,i}(x) + f'(x_i) \hat{H}_{n,i}(x) \right]\ , \tag{5.41}$$

*con* $H_{n,i}(x)$ *e* $\hat{H}_{n,i}(x)$ *polinomi di Hermite di grado* $n$, *definiti come*

$$\begin{aligned} H_{n,i}(x) &= \left[1 - 2(x - x_i) L'_{n,i}(x_i)\right] L^2_{n,i}(x) \\ \hat{H}_{n,i}(x) &= (x - x_i) L^2_{n,i}(x) \end{aligned}$$

*ove* $L_{n,i}(x)$ *è l'i-esimo polinomio di Lagrange di grado* $n$.

*Implementare (5.41) è computazionalmente costoso. Si può alternativamente usare lo schema di interpolazione nella forma di Newton nel seguente modo. Si considerano i punti*

$$\{z_0, z_1, z_2, z_3, ..., z_{2n+1}\} = \{x_0, x_0, x_1, x_1, ..., x_n, x_n\}$$

*e i corrispondenti valori della funzione $f$ e della sua derivata prima $f'$ nei punti, cosicchè il polinomio $H_{2n+1}(x)$ si può scrivere nella* **forma equivalente**

$$\begin{aligned} H_{2n+1}(x) &= q_{0,0} + q_{1,1}(x-x_0) + q_{2,2}(x-x_0)^2 + q_{3,3}(x-x_0)^2(x-x_1) \\ &+ q_{4,4}(x-x_0)^2(x-x_1)^2 + \\ &+ \cdots + q_{2n+1,2n+1}(x-x_0)^2(x-x_1)^2\cdots(x-x_{n-1})^2(x-x_n)\ . \end{aligned} \tag{5.42}$$

*Il problema è quindi ricondotto a determinare i coefficienti $q_{i,i}$, come per lo schema di interpolazione di Newton. In pratica $q_{0,0} = f[z_0]$, $q_{1,1} = f[z_0, z_1]$ ecc..*

*Sia* **Algoritmo 1** *lo schema alle differenze divise che calcola i suddetti coefficienti, restituendo l'array $(q_{0,0}, q_{1,1}, ...., q_{2n+1,2n+1})$.*

*Scrivere un programma Matlab/Octave che implementa l'***Algoritmo 1** *e costruisce il polinomio di interpolazione di Hermite mediante (5.42). Si consideri $f(x) = \sin(e^x - 2)$, $x \in [0, 2]$.*

*Produrre anche il grafico di $f$ e del polinomio interpolante.*

*Qual è il massimo errore tra la funzione e l'interpolante? L'errore ha analogie con l'errore d'interpolazione classico (solo sui dati)?*

## 5.7 Funzioni Spline

Per avere un quadro completo dell'argomento, che richiederebbe una trattazione molto più estesa, rinviamo il lettore interessato alla fondamentale monografia di Carl de Boor [8] oppure al più recente e compatto volume ad opera dell'autore [9].

**Definizione 22.** Si dice che $s$ è una funzione spline di grado $k$ se oltre ad essere un polinomio di grado $k$ è $\mathcal{C}^{k-1}[a,b]$. In tal caso i punti $x_i,\ i = 1, ..., n-1$ vengono detti *nodi* (interni).

Notazione: $\mathcal{S}(k;\, x_0, x_1, ..., x_n)$ è lo spazio lineare delle splines di grado $k$.

Una spline si può scrivere

$$s(x) = \sum_{j=0}^{k} c_j x^j + \frac{1}{k!} \sum_{j=1}^{n-1} d_j (x - x_j)_+^k,\ x \in [a,b]. \tag{5.43}$$

La funzione $(x - x_j)_+^k$ si chiama **potenza troncata** ed è definita come segue:

$$(x - x_j)_+^k = \begin{cases} (x - x_j)^k & x > x_j \\ 0 & x \le x_j \end{cases} .$$

In (5.43) ci sono $k + n$ parametri ($c_j$ e $d_j$), ciò implica che lo spazio delle splines di grado $k$ ha dimensione $n + k$.

Esempio 38. *Le splines cubiche, che sono anche le più usate, si ottengono per $k = 3$. Il comando Matlab/Octave* `spline` *costruisce proprio splines cubiche. Vedremo nel paragrafo 5.7.3 come costruire splines cubiche interpolanti imponendo diverse condizioni sui nodi di "bordo".*

`Ordine di approssimazione`: se $f \in \mathcal{C}^{k+1}[a,b]$ e se $n$ (numero nodi) è variabile, allora si prova che

$$\min_{s \in \mathcal{S}(k;\xi_0,\xi_1,...,\xi_n)} \|f - s\| = \mathcal{O}(h^{k+1})$$

con $h = \max_{1 \le i \le n-1} |x_{i+1} - x_i|$.

### 5.7.1 B-Splines

Sia $\{x_1, x_2, ..., x_n\}$ (o $\{x_i\}_{i=-\infty}^{+\infty}$) una sequenza finita (o infinita) crescente di numeri reali ($x_i < x_{i+1}$), detti *nodi* che per ora assumiamo **distinti**.

**Definizione 23.** *La i-esima B-Spline di ordine k, che si indica con $B(x; x_i, ..., x_{i+k})$ (grado $k-1$) è la k-esima differenza divisa della potenza troncata $p(x;t) = (x-t)_+^{k-1}$*

$$B(x; x_i, ..., x_{i+k}) = (x_{i+k} - x_i)p[x_i, ..., x_{i+k}](x) \ ,$$

*dove con $p[\cdot](x)$ si è indicata la k-esima differenza divisa costruita sui nodi $x_i$, $x_{i+1}$, ..., $x_{i+k}$ di $p(x;\cdot)$ vista come funzione di $x$.*

**Nota**. Per capire meglio questa definizione, suggeriamo di costruirsi le B-splines lineari, ovvero per $k = 2$.

**Proposizione 12. Alcune proprietà delle B-Splines**.

- $B_{i,k}(x) = 0$ *se* $x \notin (x_i, x_{i+k}]$.
- $B_{i,k} > 0$ *nel suo supporto* $[x_i, x_{i+k})$
- $\forall x \in \mathbb{R}$, $\sum_{i=-\infty}^{\infty} B_{i,k}(x) = 1$ *o equivalentemente*

$$\int_{\mathbb{R}} B_{i,k}(x)dx = 1 \ .$$

Le B-splines sono quindi *a supporto compatto*, *positive* e formano una *partizione dell'unità*.

**Relazione di ricorrenza**. Si basa sulla *regola di Steffensen* per la differenza divisa del prodotto di due funzioni $f$ e $g$.

**Proposizione 13. Regola di Steffensen**
*Siano $f$ e $g$ due funzioni sufficientemente differenziabili e i punti $x_1 \le ... \le x_{n+1}$ siano dati. Allora*

$$(f \cdot g)[x_1, ..., x_{n+1}] = \sum_{j=1}^{n+1} f[x_1, ..., x_j]g[x_j, ..., x_{n+1}] \tag{5.44}$$

Pertanto, se riscriviamo la funzione potenza $p(x;t) = (x-t)_+^k$ come il prodotto delle funzioni $f(x) = (x-t)$ e $g(x) = (x-t)_+^{k-1}$, possiamo applicare la regola di Steffensen per ottenere la seguente relazione di ricorrenza (utile ai fini computazionali!) per le B-splines

$$B_{i,l}(x) = \left(\frac{x_{i+l} - x}{x_{i+l} - x_i}\right) B_{i+1,l-1}(x) + \left(\frac{x - x_i}{x_{i+l-1} - x_i}\right) B_{i,l-1}(x) \ . \tag{5.45}$$

dove $l$ indica l'*ordine* (=grado +1), $i$ l'indice di intervallo. La relazione si "innesca" a partire da $B_{i,1}(x) = 1$ se $x \in [\xi_i, \xi_{i+1}]$.

In Figura 5.8, sono rappresentate le B-splines di ordine 3 (quadratiche). La suddivisione su cui sono costruite ha il secondo nodo doppio e l'ultimo nodo con molteplicità pari 3.

Vedremo nel prossimo paragrafo che scegliendo i nodi multipli, le B-splines e di conseguenza la risultante spline, diventano via via meno regolari. In particolare se il nodo ha molteplicità pari all'ordine, la B-spline diventa discontinua, come in Fig. 5.8 nel caso della prima B-spline (dall'alto) costruita sui nodi $[4, 6, 6, 6]$: infatti essa risulta discontinua in 6. La

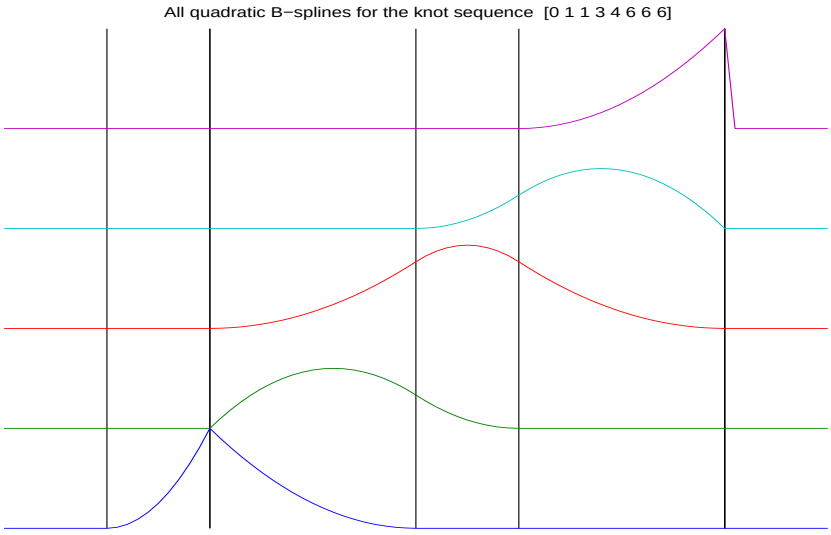


**Figura** 5.8: Bsplines di ordine 3 (quadratiche).

funzione ricorsiva `bspline.m` in Appendice C, consente di calcolare la `i`-esima B-spline di ordine `k`, data una partizione `xi` di nodi, nel generico punto `x`.

### 5.7.2 Interpolazione con funzioni spline

Sia $f(x)$ una funzione nota sui punti $t_1, t_2, ..., t_m$. Si desideri interpolarla per mezzo di una spline $S(x)$ di ordine $n$ (grado n-1) con prescritti *nodi interni* $x_1, ..., x_{N-1}$. Inoltre $t_1 < t_2 < ... < t_m$ e

$$t_1 < x_1 < x_2 < ... < x_{N-1} < t_m \, .$$

I parametri da determinare sono

$$m = N + n - 1$$

che verranno determinati dalle condizioni d'interpolazione

$$S(t_j) = f(t_j), \quad j = 1, ..., m \, . \quad (**)$$

Per l'unicità della soluzione è necessario che $\mathbf{m = N + n - 1}$ .

I. J. Schoenberg e A. Whitney nel 1953 (cfr. C. de Boor:*I.J. Schoenberg: Selected Papers, Vol 2.* pag. 342-344) hanno dimostrato che esiste *un'unica* soluzione del problema

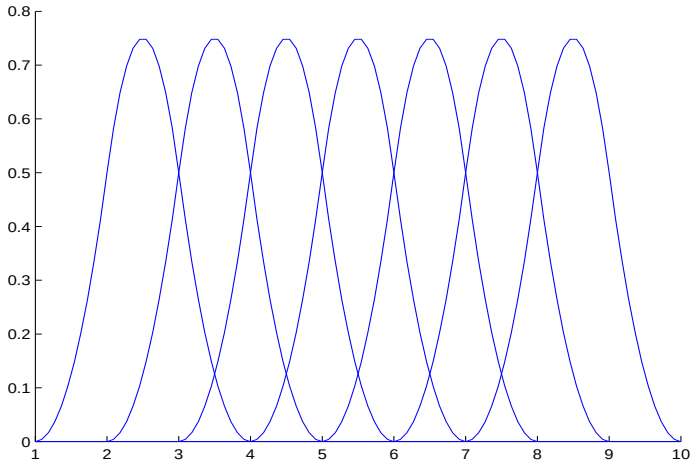


**Figura** 5.9: Bsplines quadratiche costruite con la funzione `bspline.m` sulla sequenza equispaziata `x=linspace(1,10,10)`

d'interpolazione `se e solo se` i nodi soddisfano le relazioni

$$\begin{array}{c} t_1 < x_1 < t_{n+1} \\ t_2 < x_2 < t_{n+2} \\ \vdots \quad \vdots \\ t_{N-1} < x_{N-1} < t_m \end{array} \tag{5.46}$$

**Osservazione.** Non sono richieste informazioni circa le derivate finali. In tal caso il problema d'interpolazione è trattato come un normale problema di interpolazione polinomiale.

Possiamo scrivere $S(x) = \sum_{i=1}^{m} c_i B_i(x)$ , dove $B_i$ sono B-spline di ordine $n$ con nodi **interni** la sequenza $x_1, ..., x_{N-1}$. Perciò (**) diventa

$$\sum_{i=1}^{m} c_i B_i(t_j) = f(t_j), \quad j = 1, ..., m \ . \tag{5.47}$$

ovvero, in forma matriciale, $A\mathbf{c} = \mathbf{f}$

Costruiamo le B-spline $B_i,\ i = 1, ..., m$. Aggiungiamo dapprima $2n$ nodi addizionali: $x_{1-n}, ..., x_0 \le t_1; \quad x_{1-n} < x_{2-n} < \cdots < x_0$ .

$$\begin{array}{rcl} t_m & \ge & x_N, x_{N+1}, ..., x_{N+n-1}; \\ x_N & > & x_{N+1} > \cdots > x_{N+n-1} \ . \end{array}$$

**Nota.** I $2n$ nodi addizionali possono essere presi `coincidenti` (come dimostrato da Carrasso e Laurent in *Information Processing 68*, IFIP Congress, Edinburgh 1968, Ed. A.J.H Morell, pp. 86-89).

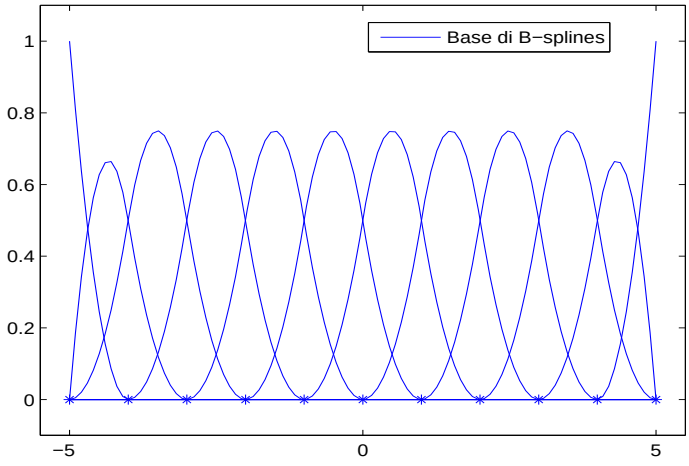


**Figura** 5.10: BSpline quadratiche costruite sulla sequenza xi = linspace(-5,5,11) con aggiunta di nodi multipli, con molteplicità pari all'ordine, agli estremi.

Per la proprietà delle B-spline di avere supporto minimo cioè

$$B_{i,n}(x) = \begin{array}{ll} & > 0 \quad x_{i-n} \le x < x_i \\ \nearrow & \\ \searrow & \\ & = 0 \quad \textit{altrimenti} \end{array}$$

si ha che la matrice $A$ ha *al più* $n$ elementi diversi da zero per ogni riga. Non solo, tale matrice è anche **stocastica** (somma x righe = somma x colonne = 1)

ESEMPIO 39. $N = 6$, $n = 4$ (spline cubiche) con nodi

$$\begin{aligned} a = t_1 \quad &< \quad t_2 < x_1 < t_3 < x_2 < x_3 < t_4 < t_5 < t_6 < x_4 < t_7 < t_8 < \\ &< \quad x_5 < t_9 = b \, . \end{aligned}$$

La matrice A $(N + n - 1)$, $9 \times 9$ sarà :

$$A = \begin{bmatrix} \times & & & & & & & & \\ \times & \times & \times & \times & & & & & \\ & \times & \times & \times & \times & & & & \\ & & & \times & \times & \times & \times & & \\ & & & \times & \times & \times & \times & & \\ & & & \times & \times & \times & \times & & \\ & & & & \times & \times & \times & \times & \\ & & & & \times & \times & \times & \times & \\ & & & & & & & & \times \end{bmatrix}$$

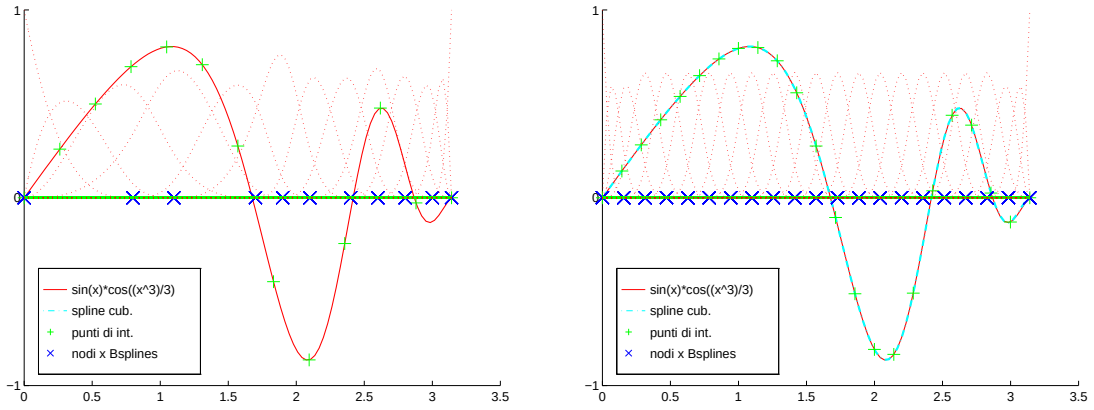


**Figura** 5.11: Spline cubica interpolante su nodi "ad hoc" a (sx) e nodi equispaziati (dx)

### 5.7.3 Interpolazione con splines cubiche

Si consideri una funzione $f(x)$ in un intervallo $I = [a, b]$ della retta reale. Si prenda una partizione di $I$ (non necessariamente equispaziata)

$$a = x_1 < x_2 < ... < x_n = b\,.$$

Facciamo vedere che il problema d' interpolazione con una spline *cubica* è equivalente alla soluzione di un sistema lineare $A \cdot m = d$ con $A$ matrice tridiagonale simmetrica, definita positiva e diagonalmente dominante.

Infatti, su ogni intervallino $I_i = [x_i, x_{i+1}]$, $i = 1, ..., n-1$ la spline è un polinomio cubico, ovvero $s_i(x) = a_{0,i} + a_{1,i}x + a_{2,i}x^2 + a_{3,i}x^3$. Per determinare univocamente la mia spline su $I$ avremo bisogno di $4(n-1)$ condizioni (tante quanti i coefficienti incogniti). Ora, le condizioni d'interpolazione negli $n$ punti $x_i$

$$s_i(x_i) = f(x_i), \quad s_i(x_{i+1}) = f(x_{i+1}), \quad i = 1, ..., n\,, \tag{5.48}$$

vanno a sommarsi alle $3(n-2)$ condizioni di continuità $\mathcal{C}^2$ nei *nodi interni*:

$$s_i(x_i) = s_{i+1}(x_i)\,,\ i = 2, ..., n-1\,, \tag{5.49}$$
$$s_i'(x_i) = s_{i+1}'(x_i)\,,\ i = 2, ..., n-1\,, \tag{5.50}$$
$$s_i''(x_i) = s_{i+1}''(x_i)\,,\ i = 2, ..., n-1\,. \tag{5.51}$$

In definitiva avremo $n + 3(n-2) = 4n - 6$. Restano pertanto *due* condizioni da assegnare per rendere univoca la determinazione dei coefficienti incogniti. Tra le possibili scelte le più usate sono le seguenti.

- $s_1''(x_1) = s_n''(x_n) = 0$, in tal caso la spline viene detta `naturale`;
- nel caso in cui siano noti i valori $f'(x_1)$ e $f'(x_n)$ s'imporranno le condizioni $s_1'(x_1) = f'(x_1)$ e $s_n'(x_n) = f'(x_n)$, in tal caso la spline viene detta `vincolata` o `completa`;
- nel caso in cui siano $f(x)$ sia periodica di periodo $x_n - x_1 = b - a$, ovvero $f(x_1) = f(x_n)$ s'imporranno le condizioni $s_1'(x_1) = s_n'(x_n)$ e $s_1''(x_1) = s_n''(x_n)$, in tal caso la spline viene detta `periodica`.

Indichiamo con $m$ il vettore dei `momenti` (cfr. eq. (5.51) )

$$\begin{aligned} m_i &= s_i''(x_i),\ i = 1, ..., n-1 \\ m_n &= s_{n-1}''(x_n)\,. \end{aligned}$$

Questa scelta riduce il numero di equazioni necessarie a determinare la spline. Infatti, $s_i''(x),\ x \in [x_i, x_{i-1}]$ è un polinomio di grado al più 1, poiché

$$s_i''(x) = m_{i+1}\frac{x - x_i}{h_i} - m_i\frac{x - x_{i+1}}{h_i} \tag{5.52}$$

dove $h_i = x_{i+1} - x_i$. Integrando due volte si ottiene

$$\begin{aligned} s_i'(x) &= m_{i+1}\frac{(x - x_i)^2}{2h_i} - m_i\frac{(x - x_{i+1})^2}{2h_i} + \alpha_i\,, \\ s_i(x) &= m_{i+1}\frac{(x - x_i)^3}{6h_i} - m_i\frac{(x - x_{i+1})^3}{6h_i} + \alpha_i(x - x_i) + \beta_i \end{aligned} \tag{5.53}$$

e le costanti $\alpha_i$ e $\beta_i$ vengono determinate imponendo le condizioni d'interpolazione nei nodi. Ovvero

$$\begin{aligned} m_i\frac{h_i^2}{6} + \beta_i &= f(x_i) \\ m_{i+1}\frac{h_i^2}{6} + \alpha_i h_i + \beta_i &= f(x_{i+1})\,. \end{aligned}$$

Da cui ricaveremo $\beta_i$, $\alpha_i$. In definitiva restano da determinare i momenti $m_i$, $i = 1, ..., n$. Dalle (5.53) imponendo la continuità della derivata prima nei nodi interni e sostituendo i valori di $\alpha_i$ e $\alpha_{i-1}$ si ottiene il sistema tridiagonale di ordine $n-2$ nelle incognite $m_1, \ldots, m_n$

$$\frac{h_{i-1}}{6}m_{i-1} + \frac{h_{i-1} + h_i}{3}m_i + \frac{h_i}{6}m_{i+1} = \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}\,, i = 2, ..., n-1\,, \tag{5.54}$$

I valori di $m_1$ e $m_n$ si determineranno imponendo le condizioni aggiuntive (5.48)-(5.51).

Vediamo ora come costruire la *spline cubica vincolata*, ovvero la spline cubica per la quale oltre ai valori della funzione $f(x)$ nei nodi devono essere soddisfatte le condizioni $s'(a) = f'(a)$ e $s'(b) = f'(b)$.

Per facilitare l'implementazione siano $x$ il vettore dei nodi della partizione, $y$ il vettorre dei valori della funzione, $y_i = f(x_i)$, i siano dati i valori aggiuntivi $y'_1 = f'(a)$ e $y'_n = f'(b)$. Pertanto l'algoritmo dovrà eseguire i seguenti passi.

1. Costruire la matrice $A$ e il vettore $d$ come segue.
   - Costruire il vettore $h$, tale che $h_i = x_{i+1} - x_i$.
   - Costruire il vettore $d$, tale che $d_1 = \frac{y_2-y_1}{h_1} - y'_1$, $d_i = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i-y_{i-1}}{h_{i-1}}$, $i = 2, ..., n-1$ e $d_n = y'_n - \frac{y_n-y_{n-1}}{h_{n-1}}$.
   - Costruire la matrice $A$, tridiagonale simmetrica, tale che
   $$A_{1,1} = \frac{h_1}{3}, \quad A_{n,n} = \frac{h_{n-1}}{3}$$
   e
   $$A_{i,i+1} = \frac{h_i}{6}, \; A_{i,i-1} = \frac{h_{i-1}}{6}, \; A_{i,i} = \frac{(h_i + h_{i-1})}{3}, \quad i = 2, ..., n-1 \; .$$
   Infine $A_{1,2} = A_{2,1}$ e $A_{n,n-1} = A_{n-1,n}$.
2. Risolvere il sistema $A\,m = d$.
3. Visualizzare i punti $(x_i, y_i)$, $i = 1, ..., n$ e la spline interpolante definita nell'intervallo $x_i \le x < x_{i+1}$, $i = 1, ..., n-1$ come segue:
   $$s(x) = \frac{(x_{i+1} - x)^3 m_i + (x - x_i)^3 m_{i+1}}{6h_i} + C\,(x_{i+1} - x) + D\,(x - x_i)$$
   dove le costanti $C, D$ sono date dalle formule seguenti: $C = \frac{y_i}{h_i} - \frac{h_i m_i}{6}$ e $D = \frac{y_{i+1}}{h_i} - \frac{h_i m_{i+1}}{6}$.

Infine, per la ricerca dell'intervallo a cui appartiene il generico punto $x$, si può fare una ricerca binaria o sequenziale. Il seguente codice Matlab/Octave esegue la ricerca sequenziale dell'indice $j$ dell'intervallo a cui appartiene il punto $x$ su cui desideriamo valutare la spline

```
function j=search_int(x,d);
%--------------------------------------------------
% Cerca l'indice j dell'intervallo a cui appartiene
% il punto x su cui valutare la spline
%--------------------------------------------------
for i=1:length(d)-1,
  if(x >= d(i) & x < d(i+1))
    j=i;
    break;
  end;
```

```
  if(x >= d(length(d)-1))
    j=length(d)-1;
    break;
  end;
end;
```

**Osservazione**. Nel toolbox `splines` di Matlab/Octave, la funzione `csapi`, (la cui chiamata si effettua come `pp=csapi(x,y)`, consente di costruire la spline cubica, nella forma definita nella struttura ppform *pp*, che soddisfa alle condizioni al bordo dette *"not-a-knot"*. Tali condizioni prevedono che nel primo nodo interno, $x_2$, e nell'ultimo interno $x_{n-1}$, sia verificata la condizione

$$\texttt{jump}\ s^{(3)}(x_2) = 0 = \texttt{jump}\ s^{(3)}(x_n)\ ,$$

### 5.7.4 Teorema del campionamento di Shannon e smoothing spline

Dato un segnale limitato in banda $s(x)$ esso può essere ricostruito dai suoi campionamenti (*Nyquist rate*) $s_k$ mediante l'uso della funzione `sinc` ovvero "sinus cardinalis", $sinc(x) = \dfrac{\sin(\pi\, x)}{\pi x}$ (forma normalizzata) oppure $sinc(x) = \dfrac{\sin(x)}{x}$ (forma non normalizzata):

$$s(x) = \sum_{k\in\mathbb{Z}} s_k \mathrm{sinc}(x-k)\ \ . \tag{5.55}$$

**Nota:** $\mathrm{sinc}(0) = 1, \mathrm{sinc}(k) = 0,\ k \in \mathbb{Z}\setminus\{0\}$. Nel caso discreto tale campionamento dà stime poco accurate.

**In alternativa**, si possono usare **splines cardinali** e relative **B-splines cardinali**. B-spline cardinali di ordine $n$ si ottengono facendo la convoluzione $n+1$ volte di $\beta^0(x) = 1, |x| < 1/2,\ \beta^0(x) = 0.5,\ |x| = 1/2$ e altrove 0. $\lim_{n\to\infty}\beta^n(x) = \mathrm{sinc}(x)$.

$$\mathbf{s(x)} = \sum_{\mathbf{k}\in\mathbb{Z}} \mathbf{s_k}\beta^{\mathbf{n}}(\mathbf{x}-\mathbf{k})\ \ .$$

Per le B-splines cardinali vale la relazione di ricorrenza

$$\beta^n(x) = \frac{x}{n-1}\beta^{n-1}(x) + \frac{n-x}{n-1}\beta^{n-1}(x-1)\ .$$

Tale scelta è più smooth e meno costosa computazionalmente.

**Smoothing**: è l'altro modo di fare *data fitting* con spline.

**Problema 1.** Siano dati i punti $(x_i, y_i),\ i = 1, ..., n$ con $y_i = f(x_i)$. Trovare la funzione $f$ che *minimizza*

$$\sum_{i=1}^{n}(y_i - f(x_i))^2 + \alpha\int_{x_1}^{x_n}(f^{(p)}(x))^2 dx\ .$$

La risultante curva è un polinomio continuo a tratti di grado $2p-1$ Il primo termine misura la vicinanza della funzione di fitting dai dati. Il secondo *penalizza* la curvatura della funzione e $\alpha$ il collegamento tra i due termini. Se $0 < \alpha < \infty$, *Schoenberg* provò che tale $f$ è la *spline naturale* di grado $2p-1$. Se $\alpha = 0$, $f$=interpolante polinomiale;

**Nota:** i dati sono assunti del tipo *segnale+rumore*

$$y_i = f(x_i) + \epsilon_i, \;\; \epsilon_i \approx N(0, \sigma^2), \; i = 1, ..., n \, .$$

## 5.8 Approssimazione con polinomi di Bernstein

Si consideri l'intervallo $[a, b] = [0, 1]$. Sia inoltre $k$ (grado) fissato. La base di B-spline sulla sequenza di nodi

$$t_0 = ... = t_k = 0, \quad t_{k+1} = ... = t_{2k+1} = 1 \, ,$$

$B_{i,k}, \; i = 0, 1, ..., k$ sono polinomi di grado $k$ su [0,1] che verificano la ricorrenza:

$$B_{i,k}(x) = xB_{i,k-1}(x) + (1-x)B_{i+1,k-1}(x) \, , \tag{5.56}$$

che è quella delle B-spline con le opportune modifiche. Sono detti *polinomi di Bernstein* di grado $k$ e si denotano con $B_i^k(x)$ o $\beta_i^k(x)$.

**Teorema 20.** *(***Teorema di Weierstrass***)*
*Sia $f \in \mathcal{C}[a, b]$. Dato $\epsilon > 0$ è sempre possibile trovare un polinomio $p_n(x)$ (di grado sufficientemente grande) tale che*

$$|f(x) - p_n(x)| \leq \epsilon, \; \forall x \in [a, b] \, .$$

**Definizione 24.** *Sia $f$ definita su $[0, 1]$. Il polinomio di Bernstein di grado $n$ associato ad $f$ è*

$$B_n(f; x) = \sum_{k=0}^{n} f(\frac{k}{n}) \binom{n}{k} x^k (1-x)^{n-k} \, .$$

**Nota:** $B_n(f; 0) = f(0), \; B_n(f; 1) = f(1)$ ( *"quasi"* interpolante).

$\beta_k^{(n)} = \binom{n}{k} x^k (1-x)^{n-k}$ *polinomi elementari di Bernstein.* Circa la convergenza dell'approssimazione di Bernstein, vale il seguente risultato

**Teorema 21.** *(di Bernstein)* Sia $f(x)$ limitata in [0,1]. Allora

$$\lim_{n \to \infty} B_n(f; x) = f(x)$$

su ogni punto $x \in [0, 1]$ dove $f$ è continua. Se inoltre $f \in \mathcal{C}[0, 1]$ allora il limite vale uniformemente.

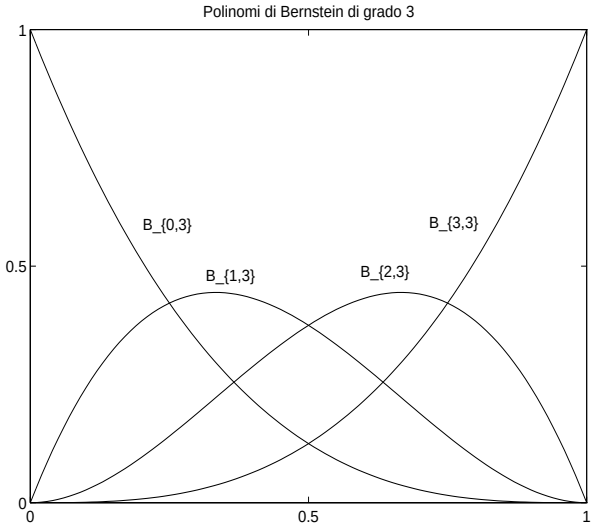


**Figura** 5.12: Polinomi di Bernstein di grado 3

Come corollario a questo teorema possiamo ri-ottenere il *Teorema di Weierstrass*.

**Corollario 1.** Se $f \in \mathcal{C}[0,1]$, allora per ogni $\epsilon > 0$ e per $n$ sufficientemente grande

$$|f(x) - B_n(f;x)| \leq \epsilon \ \ \forall x \in [0,1] \ .$$

Concludiamo con l'approssimazione con operatori di Bernstein

### 5.8.1 Curve Bspline e di Bézier

Sia $t \in [\alpha, \beta] \subset \mathbb{R}$ il parametro di una curve parametrica e $P_0, P_1, ..., P_{n-1}$, $n$ punti del piano.

1. La **curva Bspline** di ordine $m$ associata al **poligono di controllo** individuato dai punti $P_i$ è la curva
$$S(t) = \sum_{i=0}^{n-1} P_i B_{i,m}(t), \ t \in [\alpha, \beta] \ .$$

2. La **curva di Bézier** di grado $n-1$ associata al **poligono di controllo** individuato dai punti $P_i$ è la curva
$$S(t) = \sum_{i=0}^{n-1} P_i B_i^{n-1}(t), \ t \in [\alpha, \beta] \ .$$

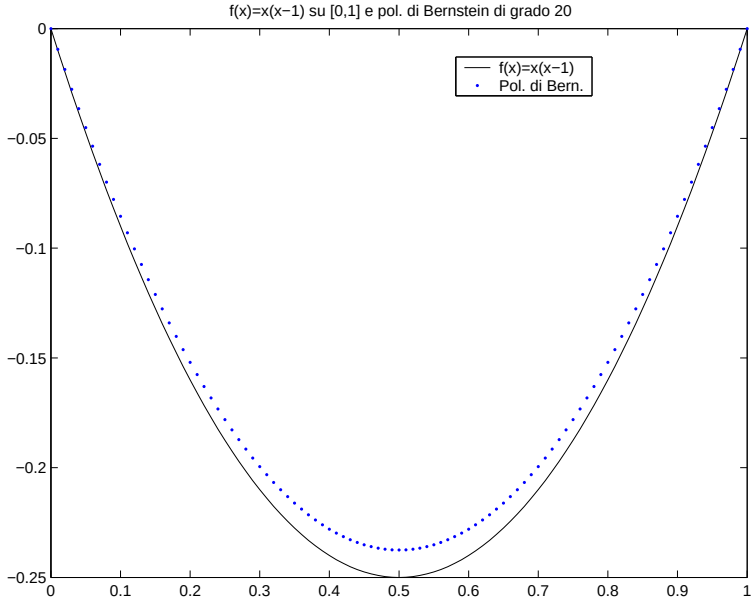


**Figura** 5.13: Approssimazione di $f(x) = x(x-1),\ x \in [0,1]$ con l'operatore di Bernstein di grado 20

$B_i^{n-1}(t)$: polinomi di Bernstein. Come per le funzioni spline, valgono gli stessi algoritmi di valutazione, derivazione e "knot-insertion". In particolare, l'algoritmo di valutazione viene chiamato di *De Casteljau*. Le cose interessanti in questo "nuovo" contesto sono relative alle *condizioni di adiacenza* tra curve di Bézier, che si esprimono come differenze finite "in avanti" dei punti di controllo, agli *algoritmi di suddivisione* e al *blossoming*. Tali curve sono però solo interpolanti agli estremi dell'intervallo (dei parametri) e approssimano il "convex-hull" della curva. Esse sono invarianti per affinità, simmetriche e "variation-diminishing".

### 5.8.2 Algoritmo di De Casteljau

I polinomi di Bernstein hanno trovato applicazione nella *geometria computazionale* e in particolare modo nella descrizione di curve e superfici di Bézier, che sono funzioni polinomiali che si ottengono con ripetute interpolazioni lineari.
Consideriamo in questa sezione un algoritmo per la costruzione di curve di Bézier noto col nome di **Algoritmo di De Casteljau** (descritto ad esempio in [10]).

**Algoritmo 4.** Dato un insieme $\mathcal{B} = \{P_0, ..., P_n\}$ di punti del piano e $t \in \mathbb{R}$ (usualmente $t \in [0,1]$), il generico punto appartenente alla curva di Bézier si determina con i seguenti passi:

1. $\{Passo\ di\ inizializzazione\}$

$$b_i^{(0)}(t) = P_i \tag{5.57}$$

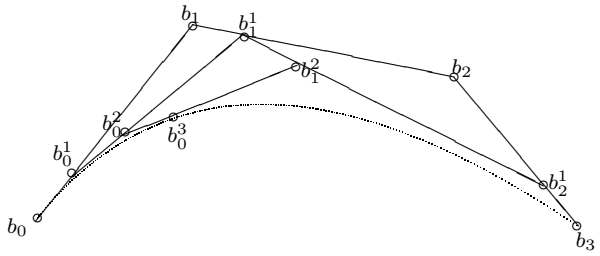


**Figura** 5.14: Costruzione di una curva di Bézier di grado 3 con l'algoritmo di De Casteljau.

*2.* $\{Passo\ iterativo\}$

$$b_i^{(r)}(t) = (1-t)b_i^{(r-1)}(t) + tb_{i+1}^{(r-1)}(t) \qquad r=1,...,n \ \ i=0,...,n-r \tag{5.58}$$

La curva di Bézier calcolata con l'algoritmo 4 è quindi ottenuta con combinazioni baricentriche ripetute. In figura 5.14 è descritto il funzionamento dell'algoritmo di De Casteljau.

**Proposizione 14.** *I punti* $b_i^{(r)}(t)$ *possono essere espressi in termini di polinomi di Bernstein* $B_j^r$ *di grado* $r$ *risultando*

$$b_i^{(r)}(t) = \sum_{j=0}^{r} P_{i+j} B_j^r(t) \quad i = 0, ..., n-r \tag{5.59}$$

**Dim.** Induzione su $r$.

$$b_i^{(r)}(t) \overset{(5.58)}{=} (1-t)b_i^{(r-1)}(t) + tb_{i+1}^{(r-1)}(t)$$
$$\overset{(5.59)}{=} (1-t)\sum_{j=i}^{i+r-1} P_j B_{j-i}^{r-1}(t) + t\sum_{j=i}^{i+r} P_j B_{j-i-1}^{r-1}(t)$$

Usiamo il fatto che $B_j^r(t) = 0$ se $j \notin \{0, ..., n\}$. Riordinando gli indici otteniamo

$$(1-t)\sum_{j=i}^{i+r} P_j B_{j-i}^{r-1}(t) + t\sum_{j=i}^{i+r} P_j B_{j-i-1}^{r-1}(t) =$$
$$\sum_{j=i}^{i+r} P_j \left[ \underbrace{(1-t)B_{j-i}^{r-1}(t) + tB_{j-i-1}^{r-1}(t)}_{B_{j-i}^r(t)} \right] = \sum_{j=i}^{i+r} P_j B_{j-i}^r(t)$$

Questo conclude la dimostrazione □

Usando l'algoritmo di De Casteljau e la geometria sottostante possiamo dedurre delle proprietà possedute dalle curve di Bézier. Di queste ricordiamo l'invarianza per affinità, l'interpolazione nei punti estremi dell'intervallo di approssimazione, la proprietà di *convex hull*[5], la simmetria e come per i polinomi di Bernstein la caratteristica capacità mimica della curva.

Un codice per costruire una curva di Bézier è il seguente

```
function [bb]=Bezier(Px,Py,t)
%-----------------------------------------------------------
% Dati i vettori Px e Py, ascisse e
% ordinate del poligono di controllo rispettivamente,
% e il vettore t dei parametri (di lunghezza m),
% la funzione costruisce la curva di Bezier
% e la salva nella matrice bb (di dimensione 2 x m)
%-----------------------------------------------------------
n=length(Px);m=length(t) b=[Px; Py];
for k=1:m
for r=2:n,
    for i=1:n-r+1,
        b(:,i)=(1-t(k))*b(:,i)+t(k)*b(:,i+1);
    end
end
bb(:,k)=b(:,1);
end

return
```

[5]Si definisce *convex hull* l'insieme formato dalle combinazioni convesse di un insieme di punti (di uno spazio euclideo) detto il *poligono di controllo*.

## 5.9 Minimi quadrati discreti e decomposizione SVD

In corrispondenza a punti (nodi) $x_i$ e ai valori $y_i$, $i = 0, ...., m$ provenienti, ad esempio, da misurazioni sperimentali, il problema dei minimi quadrati discreti consiste nell'approssimare tali valori con una funzione

$$s_n(x) = \sum_{i=0}^{n} c_i \varphi_i(x) \ , \quad n \ll m \tag{5.60}$$

ovvero una combinazione lineare di $n+1$ funzioni linearmente indipendenti $\{\varphi_i\}$, possibilmente facili da costruire. Per determinare $s_n$ si chiederà che risulti *minimo* il residuo (quadratico)

$$E(s_n) = \sum_{i=0}^{m} (s_n(x_i) - y_i)^2 \tag{5.61}$$

La scelta delle $\varphi_i$ viene dettata di solito da informazioni note sulla distribuzione dei dati o semplicemente dall'analisi grafica della loro distribuzione. Una scelta semplice è di prendere $\varphi_i(x) = x^i$, $i = 0, \ldots, n$. In tal caso il problema si riconduce alla costruizione di un polinomio di approssimazione.

Dati $m+1$ punti $(x_i, y_i)$, $i = 0, ..., m$, ci si propone di trovare un *polinomio* di grado $n \leq m$ (possibilmente $n \ll m$) t.c. siano `minime` le deviazioni (errori) $(p(x_i) - f_i)^2$, $i = 0, ..., m$.

La soluzione si ricerca minimizzando il seguente funzionale quadratico rispetto a tutti i polinomi $p$ di grado $m$

$$E(p) = \sum_{i=0}^{m} (p(x_i) - y_i)^2 = \sum_{i=0}^{m} \left\{a_0 + a_1 x_i + \cdots + a_n x_i^n - y_i\right\}^2 \ . \tag{5.62}$$

In effetti, il funzionale $E(p)$ dipendente dai coefficienti del polinomio $p$, cioè $a_0, ..., a_n$, pertanto potremo scrivere $E(a_0, ..., a_n)$ per indicarne appunto tale dipendenza. Essendo un funzionale quadratico, il minimo lo si ricerca tra i punti che annullano le derivate parziali rispetto ai coefficienti. Vale infatti il seguente Teorema

**Teorema 22.** *Condizione necessaria affinchè si raggiunga il minimo è che*

$$\frac{\partial E}{\partial a_j} = 0, \quad j = 0, ..., n \ . \tag{5.63}$$

Questo da origine al sistema

$$\sum_{i=0}^{m} \left\{a_0 + a_1 x_i + \cdots + a_n x_i^n - y_i\right\} x_i^j = 0, \quad j = 0, ..., n \, ,$$

che in forma matriciale diventa

$$\begin{pmatrix} \sum_{i=0}^{m} x_i^0 & \sum_{i=0}^{m} x_i & \cdots & \sum_{i=0}^{m} x_i^n \\ \sum_{i=0}^{m} x_i & \sum_{i=0}^{m} x_i^2 & \cdots & \sum_{i=0}^{m} x_i^{n+1} \\ \vdots & \vdots & & \vdots \\ \sum_{i=0}^{m} x_i^n & \sum_{i=0}^{m} x_i^{n+1} & \cdots & \sum_{i=0}^{m} x_i^{2n} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{m} x_i^0 y_i \\ \sum_{i=0}^{m} x_i y_i \\ \vdots \\ \sum_{i=0}^{m} x_i^n y_i \end{pmatrix} \tag{5.64}$$

Il sistema (5.64) rappresenta le cosidette **equazioni normali** e si può scrivere compattamente come

$$B\mathbf{a} = \mathbf{f} \ ,$$

con $B$ matrice simmetrica e semidefinita positiva di ordine $(n+1)$ di elementi $b_{ij} = \sum_{i=0}^{m} x_i^{i+j-2}$ e $\mathbf{z}$ vettore colonna $z_i = \sum_{j=0}^{m} x_j^{i-1} y_j$ oppure, come vedremo più oltre, ricorrendo alla decomposizione $SVD$ della matrice rettangolare $A$ i cui elementi sono $a_{i,j} = x_i^{j-1},\ i = 1, ..., m+1,\ \ j = 1, ..., n+1$.

Il seguente teorema garantisce l'esistenza e unicità della soluzione del problema dei minimi quadrati.

**Teorema 23.** *Se i punti $x_0$, ..., $x_m$ sono distinti e $n \leq m$ allora esiste ed è unico il polinomio $p$, $deg(p) \leq n$ tale che $E(p)$ è minimo. I coefficienti $a_0, \ldots, a_n$ sono determinati dalla soluzione del sistema (5.64).*

### 5.9.1 Equivalenza tra sistema dei minimi quadrati e decompozione SVD

Sia $A$ una `matrice rettangolare` $m \times n,\ m \geq n$ che rappresenta la matrice di un problema di approssimazione dell' insieme di valori $X = \{(x_i, y_i),\ i = 1, ..., m\}$ con polinomi di grado $\leq n-1$, ovvero

$$\sum_{i=1}^{n} a_i x_j^{i-1} = y_j, \quad j = 1, ..., m \ . \tag{5.65}$$

Sappiamo che $A^T A$ è $n \times n$ simmetrica e semidefinita positiva. Usando, ad esempio, il *metodo di Jacobi* per il calcolo di **tutti** gli autovalori di $A^T A$ possiamo determinare una matrice ortogonale $U$ e una matrice diagonale $D$ tale che

$$U^T (A^T A) U = D \ . \tag{5.66}$$

Ora, essendo $D = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, le cui componenti sono gli autovalori di $A^T A$ in ordine decrescente. Se qualche $\lambda_i$ risultasse un numero negativo (di modulo molto piccolo), lo si

può considerare zero, poiché gli autovalori di $A^TA$ sono tutti positivi a meno di errori di arrotondamento dovuti al metodo di calcolo (premoliplicazione di $A$ per la sua trasposta) e alla precisone usata.

Da (5.66), posto $B = AU$ $(m \times n)$, si ha che

$$B^TB = D \ . \tag{5.67}$$

Il che implica che le colonne di $B$ sono ortogonali.

Usando invece la fattorizzazione `QR` della matrice rettangolare $B$, determineremo una matrice ortogonale $V$ tale che

$$V^TB = R \tag{5.68}$$

con $R$ che è zero sotto la diagonale principale. Inoltre, la matrice $R$ è tale che

$$R^TR = B^TV^TVB = B^TB = D \ ;$$

Questo fatto ci suggerisce che le colonne di $R$ sono ortogonali. Inoltre, se per qualche $i$ si ha $\lambda_i = 0$ allora è facile verificare che la corrispondente colonna di $R$ sarà zero.

Poiché $R$ è triangolare superiore ed è ortogonale, allora essa risulta essere zero anche sopra la diagonale principale. In definitiva $R$ è diagonale ed ha la stessa forma della matrice $F$ della `decomposizione SVD` di $A$ (ricordiamo che $V^TAU = F$) cioè

$$R = \begin{pmatrix} \mu_1 & 0 & 0 & 0 & 0 \\ 0 & \mu_2 & 0 & 0 & 0 \\ \vdots & & \ddots & & \\ 0 & \cdots & & 0 & \mu_n \end{pmatrix} \tag{5.69}$$

Ora avremo che $R = F$ con $\mu_i = \sqrt{\lambda_i}$.

In (5.68), essendo $B = AU$, si ha che la decomposizione SVD richiesta è:

$$V^TAU = R \ .$$

Riassumendo, potremo dire che il vantaggio e lo svantaggio di questo approccio sono rispettivamente

1. **vantaggio:** semplicità di implementazione del metodo, una volta risolto il problema della ricerca degli autovalori di $A^TA$;
2. **svantaggio:** si deve fare il prodotto $A^TA$ che come noto può portare ad una perdita di informazioni ed ad un aumento del numero di condizionamento di $A$.

**Un esempio**

Si vuole determinare la funzione che approssima, nel senso dei minimi quadrati i punti $\{(x_i, y_i),\ 1 \le i \le m\}$, con un polinomio cubico $p_3(x) = a_1 + a_2x + a_3x^2 + a_4x^3$. Questo è quello che in inglese si chiama *"data fitting"*.

Per determinare i coefficienti $a_i,\ i = 1, ..., 4$ minimizzeremo, invece dell'errore, l'`errore quadratico medio`

$$E(a_1, a_2, a_3, a_4) = \left( \frac{1}{m} \sum_{j=1}^{m} (y_j - p_3(x_j))^2 \right)^{\frac{1}{2}} .$$

Osserviamo che minimizzare $E$ o $E^2$ è la stessa cosa. Ora, poichè $E^2$ è una funzione convessa, il minimo lo ricercheremo chiedendo che $\frac{\partial E^2}{\partial a_i} = 0,\ \ i = 1, 2, 3, 4$. Ciò dà luogo ad un sistema lineare $A\mathbf{a} = \mathbf{y}$ con $A$, $m \times 4$ e i vettori **a, y** che sono $4 \times 1$ e $m \times 1$, rispettivamente.

Vediamo il tutto in un caso concreto.

```
Si considerino i punti:

t=0:.05:1.0;
y=[.486; .866; .944;
   1.144; 1.103; 1.202;
   1.166; 1.191; 1.124;
   1.095; 1.122; 1.102;
   1.099; 1.017; 1.111;
   1.117; 1.152; 1.265;
   1.380; 1.575; 1.857];
```

Una possibile implementazione in Matlab/Octave della *decomposizione SVD* di una matrice $A$, applicata al problema dei minimi quadrati per *data fitting*, è come segue.

```
function [x]=SVD_MinQuad(t,n)
%---------------------------------------
% t=vettore dei punti
% n=grado del polinomio d'approssimazione
%
% x=soluzione del sistema con SVD
%---------------------------------------
m=length(t);

for i=1:m,
   a(i,:)=t(i).^(0:n);
end;
```

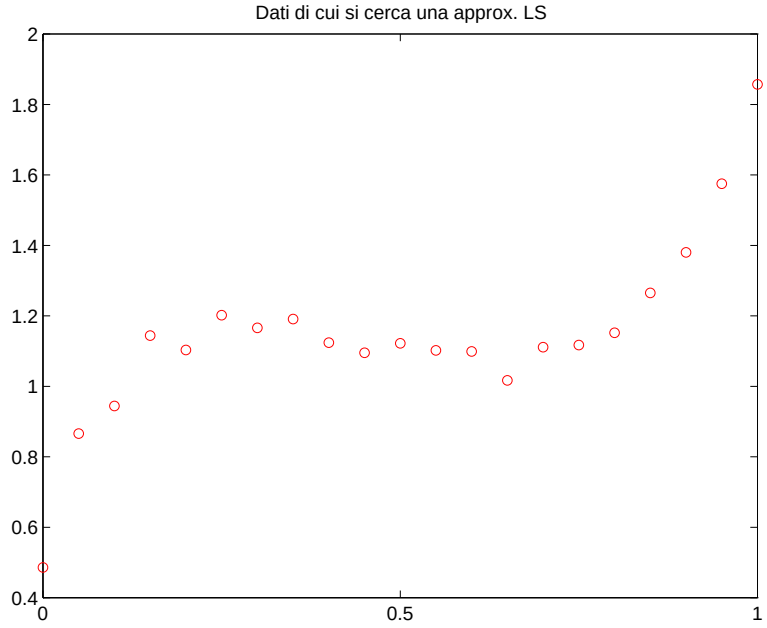


**Figura** 5.15: Dati da approssimare con il metodo dei minimi quadrati

```
[u,d]=eig(a'*a);
b=a*u;
[v,r]=qr(b);
z=inv(r'*r)*r'*(v'*y);
disp('Soluzione con la decomposizione SVD di A ');
x=u*z
```

Eseguendo il codice ecco i risultati.

```
>> Soluzione con la decomposizione SVD di A

    0.5747
    4.7259
  -11.1282
    7.6687
```

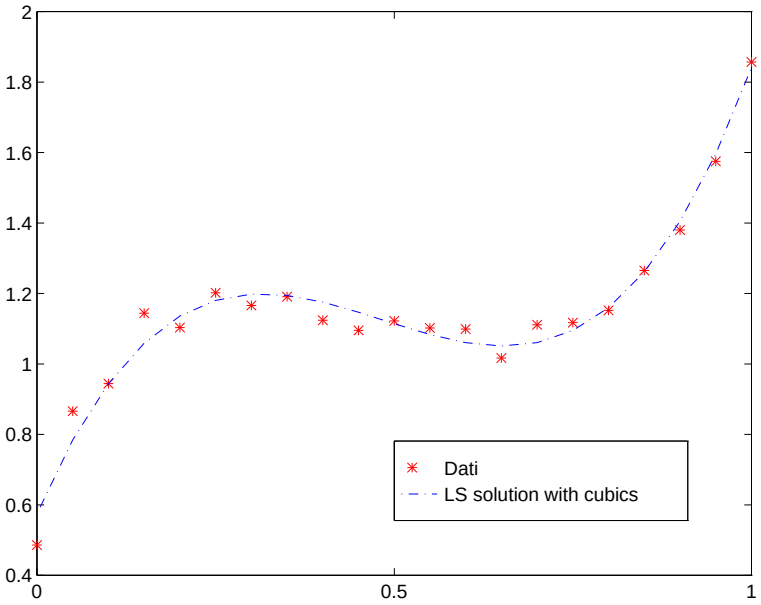


**Figura** 5.16: Approssimazione ai minimi quadrati

### 5.9.2 SVD in Matlab/Octave

In questa sottosessione, facciamo vedere come usare Matlab/Octave per determinare la decomposizione SVD *(Singular Value Decomposition)* di una matrice. Tale decomposizione, fattorizza una matrice $A \in \mathbb{R}^{n\times m}$ (anche rettangolare) nel prodotto

$$A = USV^T, \;\; U \in \mathbb{R}^{n\times n}, \;\; S \in \mathbb{R}^{n\times m}, \;\; V \in \mathbb{R}^{m\times m}$$

Gli elementi nella "diagonale" di $S$ sono chiamati *valori singolari* di $A$. Tutti gli altri elementi di $S$ sono nulli. Le matrici $U$ e $V$ sono *ortogonali*, cioè $U^TU = I_n$ e $V^TV = I_m$, ove $I_n$ e $I_m$ indicano le matrici identità di ordine $n$ e $m$, rispettivamente. Il comando `svd` calcola appunto la decomposizione SVD. Vediamo un esempio.

```
>> A=[1,2,1;1,3,1;0,1,1;1,1,1]

A =

     1     2     1
     1     3     1
     0     1     1
     1     1     1
```

```
>> [U,S,V]=svd(A)

U =

   -0.5346    0.1091   -0.1889   -0.8165
   -0.7186   -0.5603   -0.0547    0.4082
   -0.2738    0.2608    0.9258    0.0000
   -0.3505    0.7786   -0.3230    0.4082


S =

    4.5732         0         0
         0    0.7952         0
         0         0    0.6736
         0         0         0


V =

   -0.3507    0.4117   -0.8411
   -0.8417   -0.5323    0.0903
   -0.4105    0.7397    0.5332
```

A partire dai fattori $U$, $S$ e $V$ è possibile calcolare, ancora, la soluzione ai minimi quadrati di un sistema sovradeterminato

```
>> b=[1;2;0;1]

b =

     1
     2
     0
     1

>> d=U'*b

d =

   -2.3223
   -0.2329
   -0.6213
```

```
    0.4082

>> s=diag(S)

s =

    4.5732
    0.7952
    0.6736
length


>> y=d(1:length(s))./s

y =

   -0.5078
   -0.2929
   -0.9224

>> x=V*y

x =

    0.8333
    0.5000
   -0.5000
```

In maniera analoga, è possibile calcolare la soluzione ai minimi quadrati di un sistema *quadrato* singolare.

```
>> A=[1,2,1;1,2,1;0,1,0]

A =

     1     2     1
     1     2     1
     0     1     0

>> b=[1;2;0]

b =
```

```
     1
     2
     0

>> [U,S,V]=svd(A)

U =

   -0.6873   -0.1664   -0.7071
   -0.6873   -0.1664    0.7071
   -0.2353    0.9719         0


S =

    3.5616         0         0
         0    0.5616         0
         0         0         0


V =

   -0.3859   -0.5925   -0.7071
   -0.8379    0.5458         0
   -0.3859   -0.5925    0.7071
```

Il terzo valore singolare vale 0 e dunque non è possibile calcolare $y_3 = d_3/s_3$. Basta però porre $y_3 = 0$. Si può fare automaticamente con il comando `find`:

```
>> d=U'*b

d =

   -2.0618
   -0.4991
    0.7071

>> s=diag(S)

s =

    3.5616
    0.5616
```

```
         0

>> y=zeros(size(d))

y =

     0
     0
     0

>> index=find(s~=0)

index =

     1
     2

>> y(index)=d(index)./s(index)

y =

   -0.5789
   -0.8888
         0

>> x=V*y

x =

    0.7500
   -0.0000
    0.7500
```

*Nota bene*: si procede in maniera del tutto analoga per sistemi sottodeterminati (sia quadrati che rettangolari), nel caso in cui si desideri la soluzione di norma euclidea minima, piuttosto che quella con il maggior numero di zeri. Analogamente quando si desideri la soluzione ai minimi quadrati di norma euclidea minima di sistemi singolari.

### 5.9.3 Esercizi proposti

ESERCIZIO 53. `(Appello del 21/6/06)`. *Si considerino i valori di tabella*

| $x_i$ | 1 | 2.5 | 3 | 5 | 6.5 | 8 | 9.3 |
|---|---|---|---|---|---|---|---|
| $y_i$ | 4 | 2 | 3 | 3.5 | 3.9 | 7 | 5.3 |

1. *determinare il polinomio $P_m$, di grado $m = 3$ approssimante le coppie di valori $(x_i, y_i)$ nel senso dei minimi quadrati discreti.*

2. *Si giustifichi il fatto che per $m = 6$ il polinomio è interpolante.*

3. *Si consideri il punto $\bar{x} = 4$ e come valore corrispondente $\bar{y}$, quello dell'interpolante lineare sull'intervallo [3,5]. Sia ora $|P_m(\bar{x}) - \bar{y}|$ l'errore assoluto in $\bar{x}$. Far vedere che per $m = 2$ l'errore è minimo.*

## 5.10 Interpolazione trigonometrica e FFT

**Definizione 25.** *Una funzione della forma*

$$t_M(x) = \sum_{k=0}^{M} (a_k \cos(kx) + b_k \sin(kx)) \ , \tag{5.70}$$

*si chiama un* `polinomio trigonometrico` *di grado* $M$.

Se $f : [0, 2\pi] \to \mathbb{C}$ è una funzione periodica di periodo $2\pi$ ($f(0) = f(2\pi)$), se si desidera interpolarla negli $n+1$ nodi equispaziati $x_j = \frac{2\pi j}{n}$, $j = 0, ..., n$ con $t_M(x)$ chiederemo che siano soddisfatte le condizioni

$$t_M(x_j) = f(x_j), \ \ j = 0, ..., n \ . \tag{5.71}$$

Anzitutto osserviamo che $t_M(x)$ si può scrivere come segue

- se $n$ è pari e $M = n/2$

$$t_M(x) = \frac{a_0}{2} + \sum_{k=1}^{M} (a_k \cos(kx) + b_k \sin(kx)) \ ; \tag{5.72}$$

- se $n$ è dispari e $M = (n-1)/2$

$$t_M(x) = \frac{a_0}{2} + \sum_{k=1}^{M} (a_k \cos(kx) + b_k \sin(kx)) + a_{M+1} \cos((M+1)x) \ . \tag{5.73}$$

Ricordando l'identità $\mathrm{e}^{ix} = \cos x + i \sin x$ dimostriamo ora la seguente

**Proposizione 15.**

$$t_M(x) = \sum_{k=-M}^{M} c_k e^{ikx} \ ; \tag{5.74}$$

*con*

$$\begin{cases} a_k = c_k + c_{-k} \\ b_k = i(c_k - c_{-k}), \ \ k = 0, \ldots, M \ . \end{cases} \tag{5.75}$$

**Dim.** Infatti,

$$\begin{aligned} \sum_{k=-M}^{M} c_k \mathrm{e}^{ikx} &= \sum_{k=-M}^{M} c_k(\cos(kx) + i \sin(kx)) = \\ &= \sum_{k=1}^{M} c_k(\cos(kx) + i \sin(kx)) + \sum_{k=1}^{M} c_{-k}(\cos(kx) - i \sin(kx)) + c_0 \end{aligned}$$

se $n$ è pari è facile verificare che valgono le (5.75), mentre se $n$ è dispari, osservando che $t_M(x) = \sum_{k=-(M+1)}^{(M+1)} c_k \mathrm{e}^{ikx}$, si ha che i $c_k$, $k = 0, ..., M$ sono come in (5.75) e $c_{M+1} = c_{-(M+1)} = a_{M+1}/2$. $\square$

Alla luce della precedente Proposizione, possiamo scrivere $t_M(x)$ compattamente come segue

$$t_M(x) = \sum_{k=-(M+s)}^{(M+s)} c_k \mathrm{e}^{ikx}$$

con $s = 0$ quando $n$ è pari e $s = 1$ quando $n$ dispari.

Ritorniamo al problema dell'interpolazione trigonometrica, le condizioni di interpolazione (5.71) si riscrivono come segue

$$\sum_{k=-(M+s)}^{(M+s)} c_k \mathrm{e}^{ikx_j} = f(x_j)\,, \quad j = 0, \ldots, n\,. \tag{5.76}$$

Moltiplichiamo in (5.76) a sinistra e destra per $\mathrm{e}^{-imx_j}$, $0 \le m \le n$ e sommiamo su $j$. Otteniamo

$$\sum_{j=0}^{n} \left( \sum_{k=-(M+s)}^{(M+s)} c_k \mathrm{e}^{-imx_j} \mathrm{e}^{ikx_j} \right) = \sum_{j=0}^{n} \mathrm{e}^{-imx_j} f(x_j)\,. \tag{5.77}$$

Introdotta la matrice quadrata $T$, con $t_{i,j} = \mathsf{e}^{ijh}$, $0 \le j \le n$, $k = -M - s, \ldots, M + s$, le relazioni (5.77) portano quindi alla soluzione di un sistema lineare $T\mathbf{c} = \mathbf{f}$ (con ovvio significato dei simboli introdotti). Pertanto trovare i coefficienti incogniti $c_k$ si fa con un prodotto matrice-vettore che costa, in questo caso, $(n+1)^2$. Per ottenere un algoritmo più veloce, iniziamo provando una importante proprietà delle funzioni esponenziali coinvolte.

**Lemma 2.** *Le funzioni* $\left\{\mathsf{e}^{ipx_j}\right\}$, $0 \le p \le n$ *formano un sistema ortogonale, ovvero*

$$\sum_{j=0}^{n} e^{-imx_j} e^{ikx_j} = (n+1)\delta_{k,m}, \quad 0 \le m \le n\,.$$

**Dim.** Infatti, osservando che la somma è $\sum_{j=0}^{n} \mathrm{e}^{ijh(k-m)}$ con $x_j = j\,h$, $h = 2\pi/(n+1)$.

(i) Per $k = m$ è verificata: la somma si riduce $\sum_{j=0}^{n} 1 = n + 1$.

(ii) Per $k \neq m$ osserviamo che

$$\sum_{j=0}^{n} \mathrm{e}^{ix_j(k-m)} = \frac{1 - \left(\mathrm{e}^{ih(k-m)}\right)^{n+1}}{1 - \mathrm{e}^{ih(k-m)}}$$

con numeratore che è uguale a zero poiché $\left(\mathrm{e}^{ih(k-m)}\right)^{n+1} = \mathrm{e}^{i(n+1)h(k-m)} = \cos(2\pi(k-m)) + i\sin(2\pi(k-m)) = 1$. Pertanto anche quando $k \neq m$ vale la somma. $\square$.

Alla luce del precedente Lemma, possiamo concludere che

$$c_k = \frac{1}{n+1}\sum_{j=0}^{n} \mathrm{e}^{-i\,kx_j} f(x_j)\,, \quad k = -(M+s), \ldots, M+s\,. \tag{5.78}$$

In analogia con le serie di Fourier, i coefficienti $c_k$ sono detti `trasformata discreta di Fourier` (o DFT). Ricordiamo infatti, che i coefficienti della serie di Fourier continua sono

$$\gamma_k = \frac{1}{2\pi}\int_0^{2\pi} \mathrm{e}^{-ikx} f(x)dx, \quad k \in \mathbb{N}\,.$$

Da un punto di vista computazionale, il calcolo di ogni coefficiente $c_k$ in (5.78), richiede $(n+1)$ operazioni moltiplicative. Basta infatti osservare che $c_k$, nel caso $M = \frac{n}{2}$, è il prodotto scalare dei vettori $\mathbf{f} = [f(x_0), \ldots, f(x_n)]$ e $\mathbf{e} = [\mathrm{e}^{i\frac{n}{2}jh}, \ldots, \mathrm{e}^{-i\frac{n}{2}jh}]$ che hanno lunghezza $n+1$. Complessivamente, per il calcolo di tutti i coefficienti il costo è $\mathcal{O}(n^2)$.

Ma è possibile calcolare tutti i $c_k$ in modo più efficiente mediante l'algoritmo noto in inglese col nome di *Fast Fourier Transform* o `FFT`.

### 5.10.1 Algoritmo FFT

Dato l'insieme $X = \{x_0, \ldots, x_n\}$ con $m = 2^r$, $r > 1$, poniamo $\omega_m = \mathrm{e}^{\frac{2\pi i}{m}}$ cosicché l'equivalente di $c_k$ per l'insieme $X$, è

$$d_k = \frac{1}{m}\sum_{j=0}^{n} \omega_m^{-jk} x_j\ ,\ k = 0, ..., m-1\,.$$

Posto quindi $p = 2$, $q = 2^{r-1}$ (cosicchè $pq = m$)

$$d_k = \frac{1}{p}\sum_{l=0}^{p-1} \omega_m^{-kl}\left(\frac{1}{q}\sum_{s=0}^{q-1} \omega_q^{-ks} x_{l+ps}\right)\ .$$

Posto quindi

$$e_k^{(l)} = \frac{1}{q}\sum_{s=0}^{q-1} \omega_q^{-ks} x_{l+ps}, \quad l = 0, ..., p-1\ ,\ k = 0, \ldots, m-1\ , \tag{5.79}$$

allora

$$d_k = \frac{1}{p}\sum_{l=0}^{p-1} \omega_m^{-kl} e_k^{(l)}\ ,\ k = 0, \ldots, m-1\,. \tag{5.80}$$

**Complessità**. Iniziamo con l'osservare che, in (5.79), $e_{k+q}^{(l)} = e_k^{(l)}$ perchè $\omega_q^{-q} = 1$. Pertanto, per ogni $l$, calcoleremo solo i coefficienti $e_0^{(l)}, \ldots, e_{q-1}^{(l)}$ che sono una trasformata dei valori $x_l, x_{l+p}, \ldots, x_{l+p(q-1)}$. Il costo di $\{e_k^{(l)}\}$ è quindi quello di $p$ trasformate discrete di ordine $q$. Ora, calcolando preventivamente $e_k^{(l)}$, il calcolo di $d_k$ in (5.80), richiederà $mp$ moltiplicazioni. Essendo $m = 2^r$, il calcolo di $d_k$ richiede $2m$ moltiplicazioni più il costo di valutare due trasformate discrete di ordine $q = 2^{r-1}$. Continuando, la complessità totale è:

$$2m + 2\left(2 \cdot \frac{m}{2}\right) + 2^2\left(2 \cdot \frac{m}{2^2}\right) + \cdots + 2^r\left(2 \cdot \frac{m}{2^r}\right) = \sum_{k=1}^{r} 2m = 2m\, r = 2m\, \log_2(m) < m^2\,.$$

Per maggiori dettagli vedasi [1, pag.181 e ss.].

La funzione `myFFT.m`, in Appendice C, presenta un'implementazione della FFT usando le le idee appena presentate.

# Capitolo 6

# Derivazione ed integrazione

## 6.1 Derivazione

Sia $f \in \mathcal{C}^1[a,b]$. Come possiamo approssimare $f'(\hat{x})$ in un generico punto $\hat{x} \in [a,b]$? Vediamo tre approssimazioni utili in molti casi di nostro interesse.

1. `Differenze finite in avanti:` $\Delta_a$.
   Ricordando che se $f$ è derivabile in $\hat{x}$ allora

   $$f'(\hat{x}) = \lim_{h \to 0} \frac{f(\hat{x}+h) - f(\hat{x})}{h} ,$$

   allora una prima approssimazione di $f'(\hat{x})$ si ottiene usando il rapporto incrementale:

   $$f'(\hat{x}) \approx \frac{f(\hat{x}+h) - f(\hat{x})}{h} := \Delta_a f(\hat{x}) \tag{6.1}$$

   Se $f \in \mathcal{C}^2[a,b]$ avremo

   $$f(\hat{x}+h) = f(\hat{x}) + hf'(\hat{x}) + \frac{h^2}{2} f''(\xi_{\hat{x}}) ,$$

   con $\xi_{\hat{x}} \in (\hat{x}, \hat{x}+h)$. Pertanto per l'errore avremo l'espressione

   $$f'(\hat{x}) - \Delta_a f(\hat{x}) = -\frac{h}{2} f''(\xi_{\hat{x}}) , \tag{6.2}$$

   che tende a zero come $h$. In pratica $\Delta_a f(\hat{x})$ fornisce un'approssimazione del *primo ordine* della derivata di $f$ in $\hat{x}$.

2. `Differenze finite all' indietro:` $\Delta_i$.
   Come prima, una prima approssimazione di $f'(\hat{x})$ si ottiene usando il rapporto incrementale relativamente al punto $\hat{x} - h$:

$$f'(\hat{x}) \approx \frac{f(\hat{x}) - f(\hat{x} - h)}{h} := \Delta_i f(\hat{x}) \tag{6.3}$$

   Se $f \in \mathcal{C}^2[a, b]$ avremo

$$f(\hat{x} - h) = f(\hat{x}) - h f'(\hat{x}) + \frac{h^2}{2} f''(\eta_{\hat{x}}) ,$$

   con $\eta_{\hat{x}} \in (\hat{x} - h, \hat{x})$. Pertanto per l'errore avremo un'espressione simile alle differenze finite in avanti

$$f'(\hat{x}) - \Delta_i f(\hat{x}) = \frac{h}{2} f''(\eta_{\hat{x}}) , \tag{6.4}$$

   che tende a zero come $h$. In pratica $\Delta_i f(\hat{x})$ fornisce anch'esso un'approssimazione del *primo ordine* della derivata di $f$ in $\hat{x}$.

3. `Differenze finite centrali:` $\delta$.
   Una approssimazione migliore di $f'(\hat{x})$ si ottiene usando i valori di $f$ in $\hat{x} - h$ e $\hat{x} + h$ come segue:

$$f'(\hat{x}) \approx \frac{f(\hat{x} + h) - f(\hat{x} - h)}{2h} := \delta f(\hat{x}) \tag{6.5}$$

   Infatti, se $f \in \mathcal{C}^3[a, b]$

$$f(\hat{x} + h) = f(\hat{x}) + h f'(\hat{x}) + \frac{h^2}{2!} f''(\hat{x}) + \frac{h^3}{3!} f^{(3)}(\xi_{\hat{x}}) ,$$

   con $\xi_{\hat{x}} \in (\hat{x}, \hat{x} + h)$

$$f(\hat{x} - h) = f(\hat{x}) - h f'(\hat{x}) + \frac{h^2}{2!} f''(\hat{x}) + \frac{h^3}{3!} f^{(3)}(\eta_{\hat{x}}) ,$$

   con $\eta_{\hat{x}} \in (\hat{x} - h, \hat{x})$. Sommando membro a membro e dividendo per $2h$ otteniamo

$$\frac{f(\hat{x} + h) - f(\hat{x} - h)}{2h} = f'(\hat{x}) + \frac{h^2}{12} \left( f^{(3)}(\xi_{\hat{x}}) + f^{(3)}(\eta_{\hat{x}}) \right) .$$

   Pertanto l'errore assume l'espressione

$$f'(\hat{x}) - \delta f(\hat{x}) = -\frac{h^2}{12} \left( f^{(3)}(\xi_{\hat{x}}) + f^{(3)}(\eta_{\hat{x}}) \right) , \tag{6.6}$$

   che tende a zero come $h^2$. Osserviamo anche che al tendere di $h \to 0$ anche $\xi_{\hat{x}}$) e $\eta_{\hat{x}}$) tenderanno allo stesso valore. In pratica $\delta f(\hat{x})$ fornisce un'approssimazione del *secondo ordine* della derivata di $f$ in $\hat{x}$.

Data una suddivisione regolare dell'intervallo $[a,b]$, ovvero i punti $x_k = a + kh,\ k = 0, ..., n$ con $x_n = b$, da un punto di vista implementativo le formule $\Delta_a$ si possono applicare per ogni punto eccetto il punto $b$; le formule $\Delta_i$ si possono applicare per ogni punto eccetto il punto $a$ mentre le formule centrali $\delta$ si possono applicare per ogni punto interno dell'intervallo.

Nel caso delle differenze centrali, nei punti $x_0$ e $x_n$ si usano invece le seguenti approssimzioni

$$\frac{1}{2h}[-3f(x_0) + 4f(x_1) - f(x_2)] \quad \text{in } x_0 \tag{6.7}$$

$$\frac{1}{2h}[3f(x_n) - 4f(x_{n-1}) + f(x_{n-2})] \quad \text{in } x_n\ , \tag{6.8}$$

che si ottengono calcolando in $x_0$ (rispettivamente in $x_n$) la derivata prima del polinomio d'interpolazione di grado 2 della funzione $f$.

Infatti, il polinomio di secondo grado relativo ad $x_0$, si può costruire usando i punti $x_0, x_1, x_2$ ottenendo

$$p_2(x) = f(x_0)l_0(x) + f(x_1)l_1(x) + f(x_2)l_2(x)$$

dove, al solito, $l_i(x) = \prod_{j=0, j\neq i}^{2} \frac{(x - x_j)}{(x_i - x_j)}$. Derivandolo e valutandolo in $x_0$, sapendo che $x_1 = x_0 + h$ e $x_2 = x_0 + 2h$, si ottiene la (6.7).

### 6.1.1 Un esempio

Vediamo come si comportano le approssimazioni alle differenze finite in avanti e alle differenze finite centrali nel calcolo della derivata prima della funzione $f(x) = \exp(x)$ nel punto $x = 1$. Essendo $f'(x) = \exp(x)$, il valore da approssimare è quindi $\exp(1)$.

Scriviamo quindi un codice Matlab/Octave che confronta i due metodi sopracitati per valori del passo $h$ della forma $h = 2^{-k},\ k = 1, \ldots, 50$ e ne determina anche l'errore relativo commesso. Il codice si trova nel file `mydiff.m` in Appendice C.

I grafici di Figura 6.1, mostrano come entrambi i metodi siano instabili. Quando il *passo* $h$ è troppo piccolo, l'approssimazione di entrambe peggiora invece di migliorare. Nel grafico in scala semi-logaritmica, la curva in rosso coi $'-+'$ rappresenta il primo metodo, quella in nero indicata con $'-o'$ il secondo. Osserviamo che tra i due metodi il secondo sembra avere comunque *performance* migliori.

Vediamo di giustificare questo fatto analizzando l'errore. Infatti, come dimostrato, l' errore assoluto con differenze finite in avanti $\Delta_a$ è del tipo

$$E_1 = \frac{|h|\,|f^{(2)}(\xi)|}{2},\ \xi \in I(x, x_0)$$

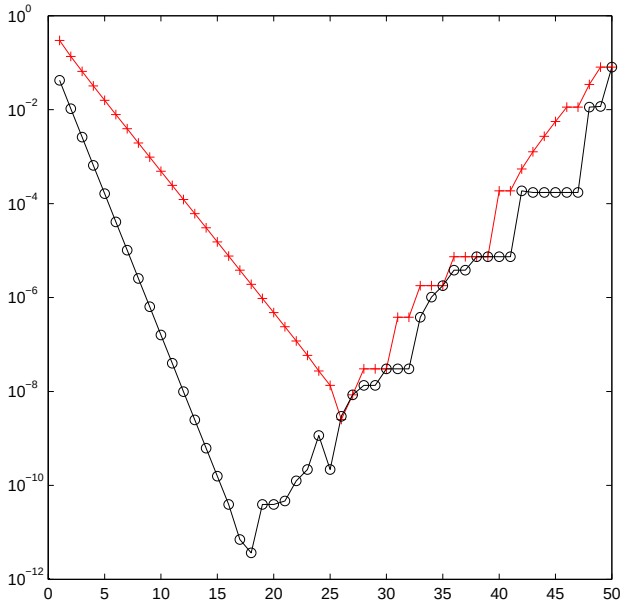


**Figura** 6.1: Grafico che illustra l'errore relativo compiuto dal metodo 1 (differenze in avanti), in rosso, col + e dal metodo 2 (differenze finite centrali) in nero con $o$, nell'approssimare $\exp(1)$.

mentre con le differenze finite centrali $\delta$ è

$$E_2 = \frac{|h|^2|\, f^{(3)}(\xi_1) + f^{(3)}(\xi_2)|}{12},\ \xi_1,\, \xi_2 \in I(x_0+h, x_0-h)$$

dove $I(s,t)$ è il più piccolo intervallo aperto contenente $s$ e $t$.

Nel nostro caso essendo $f^{(n)}(x) = \exp(x)$ per ogni $n \in \mathbb{N}$, e poiché per $x \approx 1$ si ha $\exp(x) \approx \exp(1)$ deduciamo

$$E_1 \approx \frac{|h|\exp(1)}{2} \tag{6.9}$$

$$E_2 \approx \frac{|h|^2\exp(1)}{6} \tag{6.10}$$

Per **esercizio** verificare se sono buone approssimazioni dell'errore le stime (6.9) e (6.10).

### 6.1.2 Metodi di Eulero

In questa breve sottosezione, vediamo come applicare le formule per approssimare la derivata prima alla soluzione di equazioni differenziali del primo ordine.

Consideriamo il problema di Cauchy

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \,, \end{cases} \tag{6.11}$$

con $f : I \times \mathbb{R} \to \mathbb{R}$, $t_0 \in I$. Dato l'intervallo $I = [t_0, T]$, $T < \infty$, prendiamo un passo $h = (T - t_0)/N$, con $N \geq 1$ che indica il numero dei sottointervalli in cui suddivideremo $I$, e i punti $t_n$, $0 \leq n \leq N$. Sia poi $y_n$ il valore approssimato della soluzione $y(t_n)$, ovvero $y_n \approx y(t_n)$, ottenuto con un metodo discreto per approssimare $y'(t)$

Se usiamo il rapporto incrementale in avanti $\Delta_a$

$$y'(t_n) \approx \frac{y_{n+1} - y_n}{h} \,, \tag{6.12}$$

dove $y_{n+1} = y(t_{n+1})$ e $y_n = y(t_n)$. Sostituendo in (6.11), otteniamo la formula del *metodo di Eulero esplicito (EE)*

$$y_{n+1} = y_n + h\, f_n \,, n = 0, 1, \ldots, N-1 \tag{6.13}$$

dove abbiamo usato la notazione $f_n = f(t_n, y_n)$.

Se invece dell'approssimazione (6.12) usiamo il rapporto incrementale $\Delta_i$

$$y'(t_{n+1}) = \frac{y_{n+1} - y_n}{h} \tag{6.14}$$

oppure

$$y'(t_n) = \frac{y_n - y_{n-1}}{h} \tag{6.15}$$

otterremo il *metodo di Eulero implicito (EI)* (o all'indietro)

$$y_{n+1} = y_n + h\, f_{n+1} \,, n = 0, 1, \ldots, N-1 \tag{6.16}$$

dove $f_{n+1} = f(t_{n+1}, y_{n+1})$.

Pertanto, poiché $y_0$ è nota, l'insieme dei valori $y_1, \ldots, y_N$ rappresentano la *soluzione numerica* del nostro problema.

Esempio 40. **Crescita di una popolazione**. Sia $y(t)$ una popolazione di batteri (ma questo esempio si può generalizzare al caso di una popolazione di persone) posta in un ambiente limitato, ovvero dove non possono vivere più di $B$ batteri. Sapendo che $y_0 \ll B$. Sia $C > 0$ il fattore di crescita, allora la velocità di cambiamento dei batteri al tempo $t$ sarà proporzionale al numero dei batteri presistrenti al tempo $t$, ma limiata dal fatto che non possono vivere più di $B$ batteri. L'equazione differenziale corrispondente, detta *equazione logistica*, è

$$\frac{d\, y(t)}{dt} = Cy(t) \left(1 - \frac{y(t)}{B}\right) \,, \tag{6.17}$$

che è un'equazione del primo ordine la cui soluzione mi da il numero dei batteri presenti al tempo $t$.

Se approssimiamo la derivata con il metodo di Eulero esplicito (6.13) essa diventa

$$y_{n+1} = y_n + Ch\, y_n(1 - y_n/B)\ n \geq 0\,.$$

Con Eulero implicito (6.16) essa diventa

$$y_{n+1} = y_n + Ch\, y_{n+1}(1 - y_{n+1}/B)\ n \geq 0\,.$$

In quest'ultimo caso appare evidente che usando un metodo implicito per il calcolo della soluzione al passo $t_{n+1}$, si dovrà risolvere un'equazione non lineare. Nonostante i metodi impliciti siano più costosi essi però sono più stabili (vedi, ad esempio, [19, 20]).

## 6.2 Integrazione

Si desideri calcolare l'integrale definito

$$\int_a^b f(x)dx \ .$$

I motivi che inducono a *calcolare numericamente* un integrale sono svariati: ad esempio nel caso in cui non si conosca una primitiva di $f(x)$, oppure $f(x)$ sia nota solo in alcuni punti o ancora $f(x)$ è valutabile su ogni valore di $x$ ma solo mediante una routine automatica. In tutti questi casi, si preferiscono le cosidette *formule di quadratura*. In pratica una formula di quadratura è una approssimazione dell'integrale che fa uso dei valori della funzione in alcuni punti

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} w_i f(x_i) \ , \tag{6.18}$$

dove $x_i$ sono detti *nodi di quadratura* e i coefficienti $w_i$ sono detto *pesi* della formula di quadratura.

Nel seguito ci limiteremo allo studio di integrali definiti del tipo

$$\int_a^b \omega(x)f(x)dx$$

dove $\omega(x)$ è una funzione positiva su $[a,b]$ detta *funzione peso*. Le formule di quadratura che considereremo saranno di tipo *interpolatorio* costruite sia su nodi equispaziati che su nodi coincidenti con gli zeri dei polinomi ortogonali rispetto all'intervallo $[a,b]$ e alla funzione peso $\omega(x)$.

### 6.2.1 Formule di tipo interpolatorio

Siano assegnati i punti distinti $x_0, \ldots, x_n$ dell'intervallo $[a,b]$. Sia $p_n(x) = \sum_{i=0}^n l_i(x)f(x_i)$ l'unico polinomio di grado $n$ che interpola $f$ nei punti $x_i$ ed $E_nf$ l'errore d'interpolazione. Allora, grazie alla proprietà di linearità dell'integrale

$$\int_a^b \omega(x)f(x)dx = \sum_{i=0}^{n} \left( \int_a^b \omega(x)\, l_i(x)dx \right) f(x_i) + \int_a^b \omega(x)\, E_nf(x)dx \ . \tag{6.19}$$

Posto quindi

$$w_i = \int_a^b \omega(x)\, l_i(x)dx, \quad R_nf = \int_a^b \omega(x)\, E_nf(x)dx \ ,$$

allora

$$\int_a^b \omega(x)\, f(x)dx = \sum_{i=0}^{n} w_i f(x_i) + R_nf \ . \tag{6.20}$$

Le formule di quadratura della forma (6.20) si dicono *interpolatorie* perchè si basano sul polinomio d'interpolazione della funzione $f$.

**Definizione 26.** *Una formula di quadratura di tipo interpolatorio si dice* `esatta con ordine di esattezza` $n$ *se integra esattamente i polinomi di grado* $n$.

La definizione appena data afferma che se $f(x) \in \mathbb{P}_n$ allora $E_n f(x) = 0$ e pertanto anche $R_n f = 0$. Non solo, se $f(x) = 1,\ x,\ x^2, \ldots,\ x^n$ e la formula (6.20) è esatta di ordine $n$, allora possiamo scrivere

$$\begin{cases} w_0 & + & \cdots & + & w_n & = & \displaystyle\int_a^b \omega(x)\, x^0 dx \\ w_0 x_0 & + & \cdots & + & w_n x_n & = & \displaystyle\int_a^b \omega(x)\, x dx \\ & \vdots & & \vdots & & \vdots & \\ w_0 x_0^n & + & \cdots & + & w_n x_n^n & = & \displaystyle\int_a^b \omega(x) x^n dx \end{cases} \tag{6.21}$$

dove gli integrali $\displaystyle\int_a^b \omega(x)x^k,\ k = 0, ..., n$ si chiamano *momenti*. Il sistema (6.21) è un sistema di ordine $n+1$ con matrice di Vandermonde che è non singolare poiché $x_i \neq x_j$. Pertanto il sistema può essere utilizzato per determinare univocamente i pesi $w_i,\ i = 0, \ldots, n$. L'unicità dei pesi di quadratura ci assicura anche che non esistono altre formule per i pesi che producono formule di tipo interpolatorio (6.18).

Osserviamo subito che essendo la matrice di Vandermonde *malcondizionata*, dovremmo aspettarci che per $n \to \infty$ le formule di tipo interpolatorio siano *instabili*. Vedremo in seguito come evitare questi problemi d'instabilità delle formule di tipo interpolatorio.

**Definizione 27.** *La formula di quadratura di tipo interpolatorio (6.18) si dice* `convergente` *se*

$$\lim_{n\to\infty} \sum_{i=0}^{n} w_i f(x_i) = \int_a^b \omega(x) f(x) dx\,. \tag{6.22}$$

Si può dimostrare (cfr. [4]) che se $f \in \mathcal{C}[a,b]$ si ha convergenza se

$$\sum_{i=0}^{n} |w_i| \leq C \tag{6.23}$$

con $C$ una costante *indipendente* da $n$. Ovvero si ha convergenza quando i pesi sono *limitati in modulo*. Se inoltre $f \in \mathcal{C}^k[a,b]$ si ha anche che

$$|R_n f| \leq \frac{A}{n^k}\,, \tag{6.24}$$

con $A$ costante positiva. Pertanto più $f$ è regolare e più veloce è la convergenza.

### 6.2.2 Formule di Newton-Côtes

Le formule di quadratura di Newton-Côtes, di seguito useremo N-C, sono caratterizzate dalla scelta di *nodi equispaziati*: $x_i = a + ih,\ h = (b-a)/n$. Sono di due tipi

- *formule chiuse*: quelle per cui $x_0 = a$, $x_n = b$ e $x_i = x_0 + ih$, $i = 1, \ldots, n-1$ con $h = (b-a)/n,\ n \geq 0$;
- *formule aperte*: quelle per cui $x_0 = a+h$, $x_n = b-h$ e $x_i = x_0+ih$, $i = 1, \ldots, n-1,\ h = (b-a)/(n+2),\ n \geq 0.$;

I pesi di quadratura $w_i$ delle formule di N-C hanno la caratteristica di dipendere solo da $n$ e $h$ ma *non* dall'intervallo di quadratura. Infatti, nel caso di formule chiuse e con $\omega(x) = 1$, posto $x = a + th,\ 0 \leq t \leq n$, i pesi diventano

$$w_i = \int_a^b l_i(x)dx = h\int_0^n \prod_{j=0, j\neq i}^{n} \frac{t-j}{i-j}dt\,. \tag{6.25}$$

Posti

$$\alpha_i = \int_0^n \prod_{j=0, j\neq i}^{n} \frac{t-j}{i-j}dt,\ i = 0, \ldots, n\,, \tag{6.26}$$

che dipendono solo da $i$ e $n$ ma non dai nodi $x_i$, allora la formula di quadratura diventa

$$I_n(f) = h\sum_{i=0}^{n} \alpha_i f(x_i)\ .$$

Pertanto, i "nuovi" pesi $\alpha_i$ si possono tabulare una volta per tutte usando la (6.26). Osservando che $\alpha_i = \alpha_{n-i}$, potremo limitarci a calcolarne solo la metà.

I pesi $\alpha_i$ sono noti col nome di *numeri di Côtes*. Infine, mediante la proprietà dei polinomi di Lagrange di formare una partizione dell'unità, otteniamo la relazione $\sum_{i=0}^{n} \alpha_i = n$.

Anche nel caso di formule aperte possiamo calcolare i pesi $\alpha_i$. Essendo $x_0 = a + h$, $x_n = b - h$ e $x_k = a + (k+1)h,\ k = 1, ..., n$, si ha

$$\alpha_i = \int_{-1}^{n+1} \prod_{j=0, j\neq i}^{n} \frac{t-j}{i-j}dt,\ i = 0, \ldots, n\,. \tag{6.27}$$

Nel caso particolare in cui $n = 0$, essendo $l_0(x) = 1$, da (6.27) si ha $\alpha_0 = 2$.

ESEMPIO 41. Calcoliamo i coefficienti $\alpha_i$ per le formule di N-C chiuse con $n = 1$. Essendo $x_0 = a$, $x_1 = b$, e $b - a = h$, allora

$$\alpha_0 = \int_0^1 (1-t)dt = \frac{1}{2},\ \alpha_1 = \alpha_0\ .$$

La formula di quadratura corrispondente è la ben nota `formula dei trapezi` ovvero

$$I_1(f) = \frac{h}{2}\left[f(a) + f(b)\right]\ . \tag{6.28}$$

In Figura 6.2, facciamo vedere come si comporta la formula per il calcolo di $\int_{1/2}^{2} \sin(x)\,dx$. L'area evidenziata in colore è il valore approssimato ottenuto con la formula dei trapezi. L'errore commesso è rappresentato dalla "differenza" di area tra il grafico della funzione e l'area colorata.

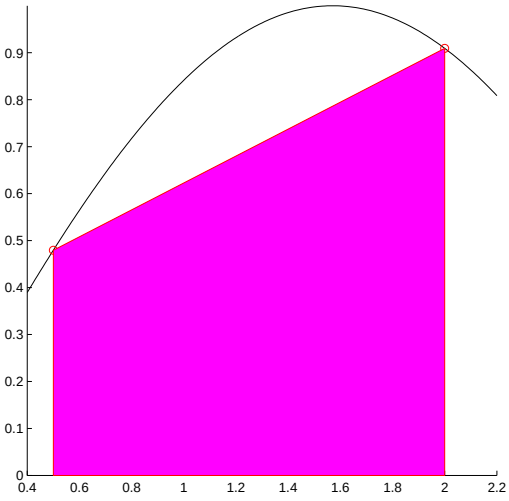


**Figura** 6.2: Regola dei trapezi per il calcolo di $\int_{1/2}^{2} \sin(x)\,dx$.

ESERCIZIO 54. *Costruire una formula di quadratura di N-C a 3 punti di tipo aperto nota come* `formula di Milne`*:*

$$\int_{-2h}^{2h} f(x)dx \approx w_1 f(-h) + w_2 f(0) + w_3 f(h)\,.$$

`Sugg.` *Determinare i pesi* $w_i$ *chiedendo l'esattezza su* $1, x, x^2$.

### 6.2.3 Stima dell'errore di quadratura

Sia $f \in \mathcal{C}^k[a,b]$. Sotto queste ipotesi di regolarità della funzione, vale il seguente risultato (la cui dimostrazione si trova ad esempio in [2, p. 721]).

**Proposizione 16.** *Al solito* $h = \frac{b-a}{n}$. *Allora*

$$R_n(f) = I_n(f) - \int_a^b f(x)dx = \gamma_n h^{k+1} \frac{f^{(k)}(\xi)}{k!},\ \xi \in (a,b)\,, \tag{6.29}$$

*con*

$$\textit{per } n \textit{ pari} \quad k = n+2 \quad \gamma_n = \int_0^n t\pi_n(t)dt$$

$$\textit{per } n \textit{ dispari} \quad k = n+1 \quad \gamma_n = \int_0^n \pi_n(t)dt$$

*dove* $\pi_n(t) = t(t-1)\cdots(t-n)$.

Riprendiamo l'esempio precedente.

- Sia $n = 1$. Essendo $n$ dispari, $k = 2$, pertanto $\gamma_1 = \int_0^1 t(t-1)dt = -1/6$. Da cui l'`errore di quadratura per la formula dei trapezi` è:

$$R_1(f) = -\frac{h^3}{6}\frac{f^{(2)}(\xi)}{2!} = -\frac{h^3}{12}f^{(2)}(\xi),\ \xi \in (a,b)\ . \tag{6.30}$$

- Sia ora $n = 2$. La funzione $f$ viene approssimata con un polinomio di secondo grado costruito usando i punti $x_0 = a$, $x_1 = \frac{a+b}{2}$ e $x_2 = b$. Pertanto

$$\alpha_0 = \int_0^2 \frac{1}{2}(t-1)(t-2)dt = \frac{1}{3},\ \alpha_1 = \int_0^2 t(2-t)dt = \frac{4}{3},\ \alpha_2 = \alpha_0\ .$$

  Da cui si ottiene la `formula di (Cavalieri)-Simpson`

$$I_2(f) = \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right]\ .$$

  Per l'errore, grazie alla Proposizione 16, otteniamo

$$R_2(f) = -\frac{1}{90}h^5 f^{(4)}(\xi),\ \xi \in (a,b)\ . \tag{6.31}$$

L'esame dell'errore di quadratura indica due situazioni

1. quando $n$ è pari, le formule di N-C sono esatte per i polinomi di grado $n+1$;

2. quando $n$ è dispari, esse sono esatte per polinomi di grado $n$.

Pertanto, ai fini dell'errore, *sono preferibili le formule per n pari*, ovvero con $n+1$ punti d'interpolazione.

Riassumiamo questi risultati in Tabella 6.1, per valori di $n = 1, \ldots, 6$.

| $n$ | $\alpha_0$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | errore |
|---|---|---|---|---|---|
| 1 | $\frac{1}{2}$ | | | | $-\frac{1}{12}h^3 f^{(2)}(\xi)$ |
| 2 | $\frac{1}{3}$ | $\frac{4}{3}$ | | | $-\frac{1}{90}h^5 f^{(4)}(\xi)$ |
| 3 | $\frac{3}{8}$ | $\frac{9}{8}$ | | | $-\frac{3}{80}h^5 f^{(4)}(\xi)$ |
| 4 | $\frac{14}{45}$ | $\frac{64}{45}$ | $\frac{24}{45}$ | | $-\frac{8}{945}h^7 f^{(6)}(\xi)$ |
| 5 | $\frac{95}{288}$ | $\frac{375}{288}$ | $\frac{250}{288}$ | | $-\frac{275}{12096}h^7 f^{(6)}(\xi)$ |
| 6 | $\frac{41}{140}$ | $\frac{216}{140}$ | $\frac{27}{140}$ | $\frac{272}{140}$ | $-\frac{9}{1400}h^9 f^{(8)}(\xi)$ |

Tabella 6.1: Formule di N-C per $n = 1, \ldots, 6$. Per $n = 1$ si ha la formula del trapezi, per $n = 2$ la formula di (Cavalieri-)Simpson e per $n = 3$ si parla di `formula dei` 3/8.

ESEMPIO 42. Vogliamo calcolare

$$I = \int_0^1 \mathrm{e}^{-x^2} dx \ ,$$

con un errore minore o uguale a $tol = 0.5 \cdot 10^{-3}$. L'integrale dato si può esprimere analiticamente mediante la `funzione errore, erf` (implementata con il nome `erf` anche in Matlab/Octave)

$$\mathtt{erf}(f)(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathtt{e}^{-t^2} dt \ ,$$

il cui grafico è riportato in Figura 6.3 ottenendo

$$I = \frac{\sqrt{\pi}}{2}\mathtt{erf}(1) \approx 0.747 \ .$$

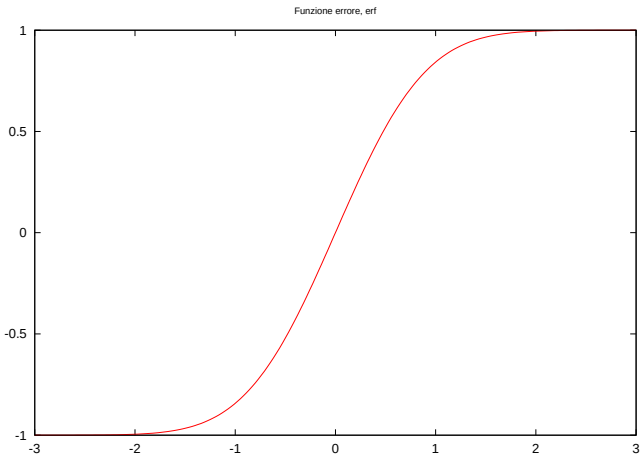


**Figura** 6.3: Grafico della funzione errore, erf

Mediante le formule date nella Proposizione 16, si tratta di trovare $n$ cosicché l'errore sia $\leq tol$.

- Partiamo con $n = 1$. Abbiamo bisogno di $\max_{x\in[0,1]} |f''(x)|$. Essendo $f''(x) = 2(2x^2 - 1)\mathrm{e}^{-x^2}$, che è strettamente crescente in $[0, 1]$ e per $x = 0$ (assume il valore minimo) vale $-2$ mentre per $x = 1$ vale $2/e < 1$. Pertanto $\max_{x\in[0,1]} |f''(x)| = 2$. Calcoliamo $\gamma_2 = \int_0^1 t(t-1)dt = -\frac{1}{6}$ e quindi, la stima richiesta, ricordando che ora $h = 1$, $k = n+1$ perchè $n$ è dispari, sarà
$$|R_1| \leq \frac{2 \cdot 1^2}{6 \cdot 2} \approx 0.166\bar{7} > tol\ .$$
Dobbiamo aumentare $n$.

- Prendiamo $n = 2$. Ragionando come prima, abbiamo ora bisogno di
$$\max_{x\in[0,1]} |f^{(4)}(x)| = \max_{x\in[0,1]} |4(4x^3 - 12x^2 + 3)\mathrm{e}^{-x^2}| = 12\ .$$
Ricordando che ora $h = 1/2$, $n = 2$, ricaviamo la stima
$$|R_2| \leq \left(\frac{1}{2}\right)^5 \cdot 12 \cdot \frac{1}{90} = \frac{2}{2880} \approx 4. \cdot 10^{-3} > tol\ .$$

- Infine prendiamo $n = 4$. Ragionando allo stesso modo, dobbiamo ora calcolare
$$\max_{x\in[0,1]} |f^{(6)}(x)| = \max_{x\in[0,1]} |8(8x^6 - 60x^4 + 90x^2 - 15)\mathrm{e}^{-x^2}| = 120\,.$$
Ricordando che ora $h = 1/4$, $n = 4$ (pari), ricaviamo la stima richiesta
$$|R_4| \leq \frac{1}{16128} \approx 6. \cdot 10^{-5} < tol\ .$$

| $n$ | $w_0$ | $w_1$ | $w_2$ | $w_3$ | $w_4$ |
|---|---|---|---|---|---|
| 8 | $\frac{3956}{14175}$ | $\frac{23552}{14175}$ | $-\frac{\mathbf{3712}}{\mathbf{14175}}$ | $\frac{41984}{14175}$ | $-\frac{\mathbf{18160}}{\mathbf{14175}}$ |

Tabella 6.2: Pesi di formule chiuse di N-C con $n = 8$

### 6.2.4 Formule di quadratura composite o generalizzate

Estendendo la Tabella 6.1 fino a $n = 8$, si può verificare che alcuni dei pesi $w_i$ risultano negativi (in Tabella 6.2 sono riportati proprio i valori dei $w_i,\ i = 0, \ldots, 4$ per le formule di N-C chiuse). Essendo alcuni di essi negativi, ciò può dar luogo ad instabilità dovuta a cancellazione numerica, rendendo pertanto le formule inutilizzabili per gradi elevati.

Una prima alternativa alle formule di Newton-Côtes classiche, sono quelle `composite` o `generalizzate`.

A tal proposito, consideriamo l'intervallo $[a, b]$ che suddividiamo in $N$ sottointervalli mediante i punti equispaziati $x_k,\ k = 0, \ldots, N$ (con $x_0 = a$ e $x_N = b$). Grazie alla additività dell'integrale possiamo scrivere

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{N-1}}^{x_N} f(x)dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x)dx \ . \quad (6.32)$$

In ciascuno dei sottointervalli $I_k = [x_k, x_{k+1}]$ applichiamo ora una formula di N-C di grado $n$. Indicato con $I_n^k(f)$ il valore dell'integrale di $f$ sul $k$-esimo intervallino $I_k$, allora

$$I(f) = \sum_{k=0}^{N-1} I_n^k(f) \ .$$

I due casi di nostro interesse sono per $n = 1$ e $n = 2$ che corrispondono alla formula dei `trapezi composita` detta anche formula `trapezoidale` e alla formula di `Simpson composita`, rispettivamente.

1. Per $n = 1$, su ogni intervallino $I_k = [x_k, x_{k+1}]$ si usa la formula dei trapezi, ovvero

$$\int_{x_k}^{x_{k+1}} f(x)dx \approx \frac{h}{2}\left[f(x_k) + f(x_{k+1})\right] \quad h = \frac{b-a}{N} \ .$$

Mettendo assieme tutti gli integrali avremo la `formula dei trapezi composita`

$$\begin{aligned} \int_a^b f(x)dx &\approx \frac{h}{2}\left[f(a) + f(x_1)\right] + \frac{h}{2}\left[f(x_1) + f(x_2)\right] + \cdots + \frac{h}{2}\left[f(x_{N-1}) + f(x_N)\right] \\ &= \frac{h}{2}\left[f(a) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{N-1}) + f(b)\right] \ . \end{aligned} \quad (6.33)$$

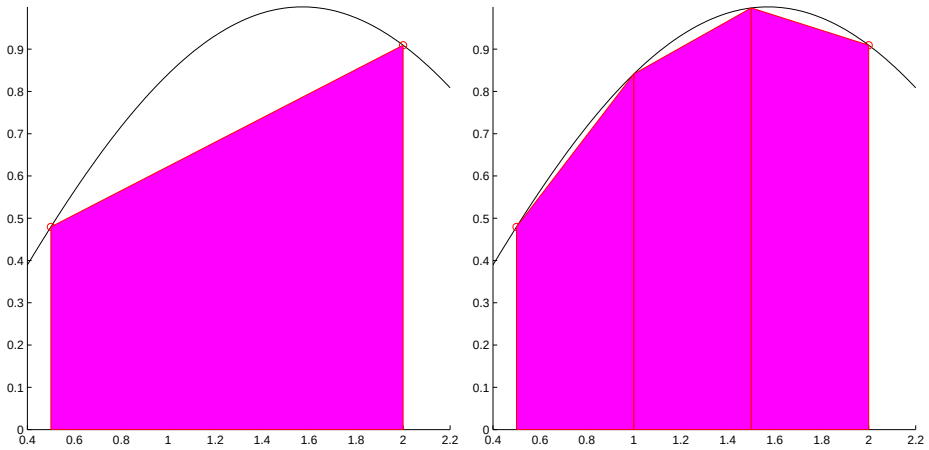

**Figura** 6.4: Confronto tra la formula dei trapezi e dei trapezi composita per il calcolo di $\int_{0.5}^{2} \sin(x)\,dx$.

2. Per $n = 2$, su ogni intervallino $I_k = [x_k, x_{k+1}]$ si usa la formula di Simpson. Ovvero,

$$\int_{x_k}^{x_{k+1}} f(x)dx \approx \frac{h}{3}\left[f(x_k) + +4f(x'_k) + f(x_{k+1})\right] \quad x'_k = \frac{x_k + x_{k+1}}{2}, \quad h = \frac{b-a}{2N}\,.$$

Osservando che su ogni intervallino $I_k$ abbiamo introdotto il punto medio $x'_k$, il che equivale a considerare i punti da $z_k,\ k = 0, \ldots, 2N$. La `formula generalizzata di Simpson` è quindi la seguente:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f(z_1) + 2f(z_2) + 4f(z_3) + \cdots + 4f(z_{2N-1}) + f(b)\right] \quad (6.34)$$

dove al solito $a = z_0$ e $b = z_{2N}$.

L'osservazione precedente sul numero dei punti si può assumere a priori e dato $N$ si considereranno sempre $2N + 1$ punti.

Come semplice esempio, in Figura 6.4 facciamo vedere come si comporta la formula dei trapezi composita rispetto a quella classica. È interessante vedere la differenza d'errore tra le due formule.

Vediamo ora come si comporta l'**errore di quadratura composita**.

Supponiamo che $f \in \mathcal{C}^s[a, b]$, sapendo che l'errore nel $k$-esimo intervallino è

$$r_n^{(k)} = \gamma_n\, h^{s+1} \frac{f^{(s)}(\xi_k)}{s!}, \quad \xi_k \in (x_k, x_{k+1}), \quad h = \frac{b-a}{N}\,,$$

allora l'errore totale sarà

$$R_n(f) = \sum_{k=0}^{N-1} r_n^{(k)} = \sum_{k=0}^{N-1} \gamma_n \, h^{s+1} \frac{f^{(s)}(\xi_k)}{s!} = \gamma_n \, \frac{h^{s+1}}{s!} \sum_{k=0}^{N-1} f^{(s)}(\xi_k) \, . \tag{6.35}$$

Si dimostra che vale la seguente uguaglianza

$$R_n(f) = \gamma_n \frac{N \, f^{(s)}(\xi)}{s!} h^{s+1} = \gamma_n (b-a)^{s+1} \frac{f^{(s)}(\xi)}{s! N^s} \quad \xi \in (a,b) \, . \tag{6.36}$$

Nei due casi precedentemente studiati, trapezi e Simpson compositi, valgono le seguenti formule d'errore:

$$R_1(f) = -\frac{(b-a)^3}{12N^2} f''(\xi) \, , \tag{6.37}$$

$$R_2(f) = -\frac{(b-a)^5}{2880N^4} f^{(4)}(\xi) \, . \tag{6.38}$$

Infine, grazie alla (6.36), ricordando che $N$ dipende in effetti da $n$, se $f \in \mathcal{C}^s[a,b]$ allora $\lim_{N\to\infty} |R_N(f)| = 0$. Ovvero, fissato $\epsilon > 0$, possiamo trovare $N$ tale che $|R_{N+1}| < \epsilon$.

ESEMPIO 43. Riprendiamo l'Esempio 42. Vogliamo approssimare $\displaystyle\int_0^1 e^{-x^2} dx$ a meno di $tol = 0.5\,10^{-3}$ con le formule composite dei trapezi e di Simpson.

- `Trapezi composito`. Sapendo che $\max_{x\in[0,1]} |f^{(2)}(x)| = 2$, si ha $|R_1(f)| \le 1/(6\,N^2)$ . Pertanto, affinché $|R_1(f)| < tol$, dovremo chiedere che $N \ge 19$, ovvero dovremo prendere 20 punti equispaziati.
- `Simpson composito`. Sapendo che $\max_{x\in[0,1]} |f^{(4)}(x)| = 12$, si ha $|R_2(f)| \le 12/(2880\,N^4)$ . Pertanto, affinché $|R_2(f)| < tol$, dovremo chiedere che $N \ge 2$, ovvero dovremo prendere 5 punti equispaziati.

### 6.2.5 Routine adattativa per la quadratura: applicazione al metodo di Simpson e dei trapezi

L'idea delle routine adattative è di usare punti di integrazione dove "serve", ovvero dove la funzione ha maggiori oscillazioni o discontinuità. La tecnica adattativa ha lo scopo di variare la posizione dei nodi secondo il *comportamento locale* della funzione integranda, avendo così un risparmio sul numero di valutazioni della funzione integranda.

In appendice C si trova la funzione `simp_ada.m` (che implementa la routine adattativa di Simpson per il calcolo di $\displaystyle\int_a^b f(x)dx$ . Come input l'utente fornirà gli estremi di integrazione

$a$, $b$, la tolleranza *epss* e gli verrà richiesto di passare un parametro di *dilatazione* della tolleranza allo scopo di rendere la stima sui sottointervalli più conservativa possibile, al fine di verificare la disuguaglianza

$$|\int_a^b f(x)dx - \tilde{I}(f)| \leq \epsilon \ ;$$

ove $\tilde{I}(f)$ è l'approssimazione dell'integrale calcolata con una formula di quadratura composita.

In output si otterranno il valore approssimato dell'integrale, nella variabile *integral* e il numero di valutazioni della funzione integranda in $nv$.

Le Figure 6.5 e 6.6 mostrano la differenza tra la routine classica e quella adattativa applicate al calcolo numerico di

$$\int_1^3 \frac{100}{x^2}\sin(\frac{10}{x})dx$$

con precisione $\epsilon = 1.e-5$. Dall'output ottenuto, il numero di valutazioni con il metodo classico è di 256 contro le 161 con la routine adattativa (avendo usato un fattore di dilatazione 15). Infine l'errore calcolato è di $2.43 \cdot 10^{-6}$ con il metodo classico contro $4.17 \cdot 10^{-7}$ con il metodo adattativo.

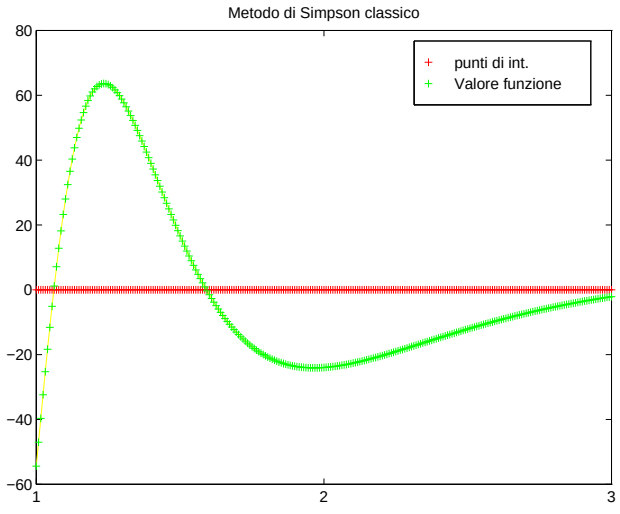


**Figura** 6.5: Integrazione con Simpson composito

Alternativamente una funzione Matlab che calcola adattativamente un integrale definito usando il metodo trapezoidale è la seguente. Si parte considerando una formula base su 3 punti e stimando l'errore usando l'*estrapolazione di Richardson* (vedi sessione 6.4 per dettagli), che sostanzialmente equivale ad usare la formula di Simpson sugli stessi punti.

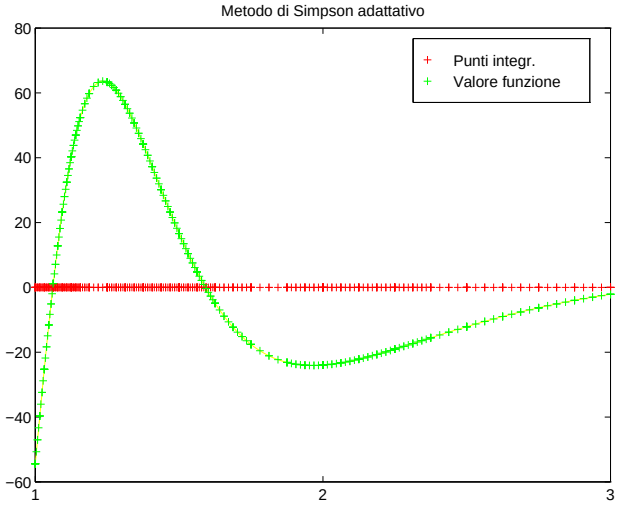


**Figura** 6.6: Integrazione con Simpson adattativo

Detto

$$\tilde{I}^{(i)} = \frac{h_i}{4}\left\{ f(x_{i-1}) + 2f(\frac{x_i + x_{i-1}}{2}) + f(x_i) \right\} ,$$

l'integrale approssimato con la formula trapezoidale su $[x_{i-1}, x_i]$ con $h_i = x_i - x_{i-1}$. Allora

$$e_i = \int_{x_{i-1}}^{x_i} f(x)dx - \tilde{I}^{(i)} \approx \frac{h_i}{12}\left\{ -f(x_{i-1}) + 2f(\frac{x_i + x_{i-1}}{2}) - f(x_i) \right\} .$$

Pertanto, se l'errore $e_i$ verifica

$$e_i \leq \frac{\epsilon}{b-a} h_i$$

(cosicchè quello totale risulta essere $\leq \epsilon$) allora si conclude altrimenti si procede alla nuova suddivisione.

La function Matlab/Octave `trap_ada.m`, in Appendice C, fa proprio questo.

La Figura 6.7 applica la routine adattativa trapezoidale appena descritta al calcolo di $\int_{-3}^{3} \frac{\sin(x)}{(1+e^x)} dx$ con precisione $\epsilon = 1.e-4$. In output si otterrà il valore approssimato dell'integrale, nella variabile `I`, l'errore approssimato in `errest` e nel vettore `x` i punti usati per il calcolo.

**Nota bibliografica**. Nel repository

`http://www.mathworks.com/matlabcentral/fileexchange/`

che raccoglie il software di scambio degli utenti Matlab, si trova il package `adaptQuad` di Matthias Conrad e Nils Papenberg che contiene due routine iterative per la quadratura

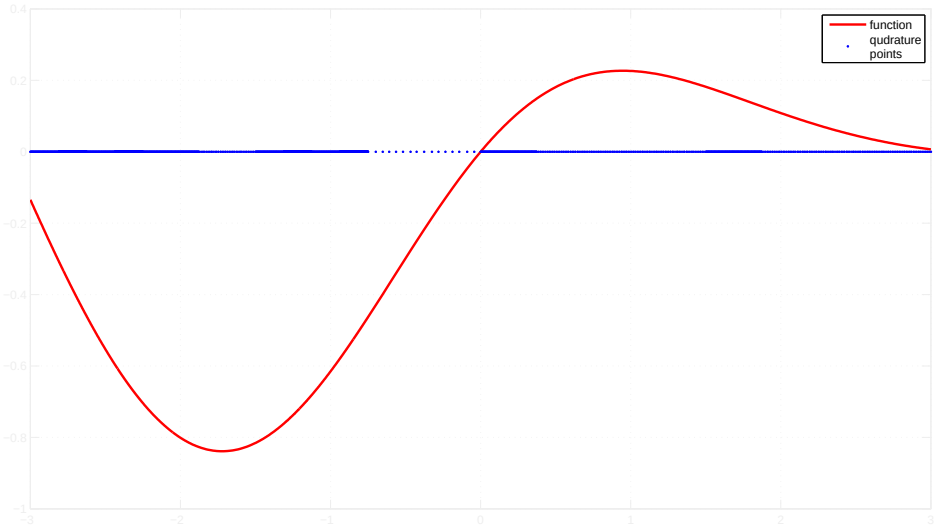


**Figura** 6.7: Integrazione con il metodo dei trapezi adattativo. I punti utilizzati sono oltre 2000, molti di più di quelli richiesti dalla stima a priori (6.37), ma distribuiti non uniformemente ma dove la funzione oscilla di maggiormente.

adattativa usando sia Simpsons che Lobatto. Per maggiori informazioni si rinvia al Techical Report [6].

### 6.2.6 Polinomi ortogonali (cenni) e formule di quadratura gaussiane

Prima di introdurre le formule di quadratura gaussiane, facciamo dei richiami sui `polinomi ortogonali.`

**Definizione 28.** *Un insieme infinito di polinomi* $\{p_0, p_1, \ldots, p_n, \ldots\}$ *tali che*

$$p_n(x) = a_{n,0}x^n + a_{n,1}x^{n-1} + \cdots + a_{n,n} \ ,$$

*è detto* **ortogonale** *in [a,b] rispetto ad una funzione peso* $\omega(x)$ *non negativa, se valgono le relazioni*

$$\begin{cases} \displaystyle\int_a^b \omega(x)p_n(x)p_m(x)dx = 0 & m \neq n \\ \displaystyle\int_a^b \omega(x)(p_n(x))^2dx > 0 & m = n \ . \end{cases}$$

Di solito si indica $h_n = \int_a^b \omega(x)(p_n(x))^2 dx > 0$.

Alcune importanti proprietà dei polinomi ortogonali sono le seguenti.

(a) La funzione peso non negativa $\omega(x)$ e l'intervallo $[a,b]$ definiscono univocamente l'insieme dei polinomi $\{p_n\}$.

(b) Per ogni $n \geq 1$, $p_n(x)$ ha esattamente $n$ zeri reali, distinti ed interni ad $[a,b]$. Inoltre gli zeri di $p_n(x)$ separano quelli di $p_{n-1}(x)$ (tra 2 zeri di $p_n$ si trova uno ed uno solo zero di $p_{n-1}$).

(c) Ogni sistema di polinomi ortogonali $\{p_n\}$, soddisfa ad una relazione di ricorrenza a 3 termini

$$p_{n+1}(x) = (A_n x + B_n)p_n(x) - C_n p_{n-1}(x), \quad n = 1, 2, \ldots \tag{6.39}$$

dove $C_n > 0$ e

$$A_n = \frac{a_{n+1,0}}{a_{n,0}} \tag{6.40}$$

$$B_n = A_n \left( \frac{a_{n+1,1}}{a_{n+1,0}} - \frac{a_{n,1}}{a_{n,0}} \right), \tag{6.41}$$

$$C_n = \frac{A_n}{A_{n-1}} \frac{h_n}{h_{n-1}} = A_n \frac{a_{n+1,2}}{a_{n+1,1}}, \tag{6.42}$$

Elenchiamo qui di seguito i polinomi ortogonali che per noi rivestono maggior interesse.

**$\mathbf{T_n}$** : `Polinomi di Chebyshev di prima specie.` Sono definiti su $[-1,1]$, $\omega(x) = (1-x^2)^{-1/2}$ e per essi vale la ricorrenza $T_0(x) = 1$, $T_1(x) = x$ e

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad n \geq 1\,. \tag{6.43}$$

Infatti, ricordando che $T_n(x) = \cos(n \arccos x)$ $n = 0, 1, \ldots$ e la relazione

$$\cos[(n+1)\theta] + \cos[(n-1)\theta] = 2\cos\theta \ \cos(n\theta)$$

posto $\theta = \cos x$ si riottiene la (6.43).

Facciamo anche vedere che

$$T_n(x) = 2^{n-1}x^n + \cdots \tag{6.44}$$

Infatti, essendo $T_2(x) = 2x^2 - 1$, $T_3(x) = 4x^3 - 3x = 2^{3-1}x^3 - 3x$, per induzione si ottiene la (6.44).

Infine, gli zeri del polinomio di Chebyshev di prima specie di grado $n$, che sono stati introdotti al capitolo dell' interpolazione polinomiale, sono i *punti di Chebyshev*

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, ..., n\,.$$

**$\mathbf{U_n}$**: `Polinomi di Chebyshev di seconda specie.` Sono definiti su $[-1,1]$, $\omega(x) = (1-x^2)^{1/2}$ e per essi vale la ricorrenza $U_0(x) = 1,\ U_1(x) = 2x$ e

$$U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x), \quad n \geq 1 .$$

**$\mathbf{P_n}$**: `Polinomi di Legendre.` Sono definiti su $[-1,1]$, $\omega(x) = 1$ e per essi vale la ricorrenza $P_0(x) = 1,\ P_1(x) = 2x$ e

$$P_{n+1}(x) = \frac{2n+1}{n+1}xP_n(x) - \frac{n}{n+1}P_{n-1}(x), \quad n \geq 1 .$$

In questo caso possiamo anche facilmente calcolare $a_{n,0} = \frac{(2n)!}{2^n(n!)^2}$ e $h_n = 2/(2n+1)$.

**$\mathbf{L_n}$**: `Polinomi di Laguerre.` Sono definiti su $[0,+\infty)$, $\omega(x) = \mathrm{e}^{-x}$ e per essi vale la ricorrenza $L_0(x) = 1,\ L_1(x) = 1-x$ e

$$L_{n+1}(x) = \frac{2n+1-x}{n+1}L_n(x) - \frac{n}{n+1}L_{n-1}(x), \quad n \geq 1 .$$

Anche in questo caso possiamo calcolare $a_{n,0} = \frac{(-1)^n}{n!}$ e $h_n = 1$.

**$\mathbf{H_n}$**: `Polinomi di Hermite.` Sono definiti su $(-\infty,+\infty)$, $\omega(x) = \mathrm{e}^{-x^2}$ e per essi vale la ricorrenza $H_0(x) = 1,\ H_1(x) = 2x$ e

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x), \quad n \geq 1 .$$

In questo caso $a_{n,0} = 2^n$ e $h_n = 2^n n!\sqrt{\pi}$.

Vale la pena osservare che in $[-1,1]$ i polinomi ortogonali di Legendre e di Chebsyshev sono un caso particolare di una famiglia più generale e associata alla funzione peso $\omega(x) = (1-x)^\alpha(1+x)^\beta,\ \alpha,\beta > -1$, detti `polinomi di Jacobi`, $P_n^{\alpha,\beta}(x)$. Posto $\gamma = \alpha+\beta$, per essi vale la ricorrenza

$$\begin{aligned} P_{n+1}^{\alpha,\beta}(x) \quad = \quad & \\ & \frac{(2n+1+\gamma)[(\alpha^2-\beta^2)+(2n+\gamma+2)(2n+\gamma)x]}{2(n+1)(n+\gamma+1)(2n+\gamma)}P_n^{\alpha,\beta}(x) + \\ & \frac{2(n+\alpha)(n+\beta)(2n+\gamma+2)}{2(n+1)(n+\gamma+1)(2n+\gamma)}P_{n-1}^{\alpha,\beta}(x), \quad n \geq 1 . \end{aligned}$$

Pertanto, per $\alpha = \beta = 0$ otteniamo i polinomi di Legendre, per $\alpha = \beta = -1/2$ otteniamo i polinomi di Chebyshev di prima specie e per $\alpha = \beta = 1/2$ otteniamo i polinomi di Chebyshev di seconda specie.

$\diamond\diamond$

Siamo ora in grado di descrivere come si possano costruire le `formule di quadratura gaussiane`. Dato l'intervallo $[a,b]$ e la funzione peso $\omega(x)$, siano $x_i,\ i=1,\ldots,n$ gli zeri del corrispondente polinomio ortogonale di grado $n$. Allora possiamo scrivere

$$\int_a^b \omega(x)f(x)dx \approx \sum_{i=1}^n A_i f(x_i) \tag{6.45}$$

dove i pesi $A_i$ dipendono dalla particolare formula di quadratura gaussiana.

Prima di dare alcune espressioni esplicite dei pesi di quadratura, enunciamo un risultato fondamentale per la quadratura gaussiana (la cui dimostrazione si può trovare, ad esempio, in [20]).

**Teorema 24.** *Siano $x_1,\ldots,x_n$ gli zeri del polinomio ortogonale di grado n rispetto all'intervallo $[a,b]$ e alla funzione peso $\omega(x)$. Supponiamo che i pesi $A_i$ siano stati determinati cosicché*

$$\int_a^b \omega(x)f(x)dx = \sum_{i=1}^n A_i f(x_i) + R_n(f)\ , \tag{6.46}$$

*è esatta per i polinomi di grado $\leq n-1$. Allora la formula (6.46) è esatta per tutti i polinomi di grado $\leq 2n-1$.*

Una caratteristica ulteriormente positiva delle formule di quadratura gaussiane, che è uno dei motivi per i quali sono preferite rispetto a quelle di N-C, è che i pesi $A_i$ `sono positivi`. Infatti vale la rappresentazione

$$A_i = \frac{1}{(P_n'(x_i))^2}\int_a^b \omega(x)\left[\frac{P_n(x)}{x-x_i}\right]^2 dx\ i=1,\ldots,n\ , \tag{6.47}$$

dove $P_n$ indica il polinomio ortogonale di grado $n$ relativo all'intervallo $[a,b]$ e alla funzione peso $\omega(x)$. Da questa relazione segue che $\int_a^b \omega(x)dx = \sum_{i=1}^n A_i = \sum_{i=1}^n |A_i|$, pertanto si ha convergenza delle formule al valore dell'integrale. Pertanto nel caso $[-1,1]$, $\omega(x)=1$, si ha $\sum_{i=1}^n A_i = 2$.

- $[a,b]=[-1,1],\ \omega(x)=(1-x^2)^{-1/2}$, la corrispondente formula di quadratura si dice di `Gauss-Chebyshev di prima specie`, GC1. I pesi sono

  $$A_i^{(GC1)} = \frac{\pi}{n},\ \forall\, i$$

  la cui somma è $\pi$. I nodi, che sono gli zeri di Chebyshev, sono

  $$x_i^{(GC1)} = \cos\left(\frac{2i-1}{2n}\pi\right)\ i=1,...,n\ .$$

La (6.45) diventa

$$\int_{-1}^{1} f(x)\frac{1}{\sqrt{1-x^2}}dx \approx \frac{\pi}{n}\sum_{i=1}^{n} f\left(\cos\left(\frac{2i-1}{2n}\pi\right)\right) .$$

- Sempre in $[-1,1]$ ma con $\omega(x) = (1-x^2)^{1/2}$: la corrispondente formula di quadratura si dice di `Gauss-Chebyshev di seconda specie`, GC2. I pesi sono

$$A_i^{(GC2)} = \frac{\pi}{n+1}\left(\sin\left(\frac{i\pi}{n+1}\right)\right)^2,\ i = 1,\ldots,n\,.$$

Siccome $\sum_{i=1}^{n}\left(\sin\left(\frac{i\pi}{n+1}\right)\right)^2 = \frac{n+1}{2}$ otteniamo il risultato richiesto $\sum_{i=1}^{n} A_i^{(GC2)} = \frac{\pi}{2}$.

I nodi, che sono gli zeri dei polinomi di Chebyshev di seconda specie, sono

$$x_i^{(GC2)} = \cos\left(\frac{i}{n+1}\pi\right)\ i = 1,...,n\ .$$

La (6.45) diventa

$$\int_{-1}^{1} f(x)\sqrt{1-x^2}dx \approx \frac{\pi}{n+1}\sum_{i=1}^{n}\left(\sin\left(\frac{i}{n+1}\pi\right)\right)^2 f\left(\cos\left(\frac{i}{n+1}\pi\right)\right) .$$

- Sempre in $[-1,1]$ ma con $\omega(x) = 1$: la corrispondente formula di quadratura si dice di `Gauss-Legendre`. I pesi sono

$$A_i = \frac{2}{(1-x_i^2)\left(P'_{n+1}(x_i)\right)^2},\ i = 0,...,n\,.$$

Riassumiamo nella Tabella 6.3, per $n = 1,\ldots,4$, i valori dei nodi (zeri) del polinomio di Legendre e dei corrispondenti pesi. Si noti che sono indicati $n+1$ nodi, poiché per un dato $n$ calcoliamo $i = 0,...,n$ nodi e pesi. Ad esempio, per $n = 1$, significa che stiamo considerando il polinomio di grado 2, che ha appunto zeri $\pm 3^{-1/2}$. Per i pesi sono indicati, per simmetria, solo quelli con $i = 0,...,\lfloor\frac{n}{2}\rfloor$. Sempre relativamente alla formula di quadratura di Gauss-Legendre, osserviamo che talvolta conviene includere anche gli estremi dell'intervallo, ovvero $-1, 1$. Si parla allora di formule di quadratura di `Gauss-Legendre-Lobatto`. Ora, i nodi $x_0 = -1$ e $x_n = 1$ sono fissati, gli altri $n-1$ sono scelti come gli zeri di $P'_n(x)$ ottenendo per i pesi l'espressione

$$A_i = \frac{2}{n(n+1)}\frac{1}{(P_n(x_i))^2},\ i = 0,...,n\ .$$

| $n$ | $x_i$ | $A_i$ |
|---|---|---|
| 1 | $\pm\frac{1}{\sqrt{3}}$ | 1 |
| 2 | $\pm\frac{\sqrt{15}}{5}$, 0 | $\frac{5}{9}$, $\frac{8}{9}$ |
| 3 | $\pm\frac{1}{35}\sqrt{525-70\sqrt{30}}$, $\pm\frac{1}{35}\sqrt{525+70\sqrt{30}}$ | $\frac{1}{36}(18+\sqrt{30})$, $\frac{1}{36}(18-\sqrt{30})$ |
| 4 | $0, \pm\frac{1}{21}\sqrt{245-14\sqrt{70}}, \pm\frac{1}{21}\sqrt{245+14\sqrt{70}}$ | $\frac{128}{225}, \frac{1}{900}(322+13\sqrt{70})$, $\frac{1}{900}(322-13\sqrt{70})$ |

Tabella 6.3: Nodi e pesi per le formule di Gauss-Legendre con $n = 1, 2, 3, 4$

| $n$ | $x_i$ | $A_i$ |
|---|---|---|
| 1 | $\pm 1$ | 1 |
| 2 | $\pm 1$, 0 | $\frac{1}{3}$, $\frac{4}{3}$ |
| 3 | $\pm 1$, $\pm\frac{\sqrt{5}}{5}$ | $\frac{1}{6}$, $\frac{5}{6}$ |
| 4 | $\pm 1$, $\pm\frac{\sqrt{21}}{7}$, 0 | $\frac{1}{10}, \frac{49}{90}$, $\frac{32}{45}$ |

Tabella 6.4: Nodi e pesi per le formule di Gauss-Legendre-Lobatto con $n = 1, 2, 3, 4$.

Pertanto, il grado di esattezza delle formule di Gauss-Legendre-Lobatto sarà $2n-1$. In Tabella 6.4 ricordiamo chi sono i nodi e i pesi per le formule di Gauss-Legendre-Lobatto con $n = 1, 2, 3, 4$. Un altro interessante esempio è fornito dalle formule di Gauss-Chebyshev-Lobatto, in cui $\omega(x) = 1/\sqrt{1-x^2}$, delle quali i nodi ed i pesi sono come segue

$$\begin{aligned} x_i &= \cos\left(\frac{\pi\, i}{n}\right), \\ A_i &= \frac{\pi}{n\, d_i}, \;\; d_0 = d_n = 2, \; d_i = 1, \; 1 \le i \le n-1\,. \end{aligned}$$

Infine, per quanto riguarda l'errore di quadratura con formule di Gauss-Legendre (GL) e Gauss-Legendre-Lobatto (GLL), ricordiamo le seguenti formule che per essere applicate richiedono una certa regolarità della funzione integranda (cfr. [19]).

$$I(f) - I_{GL}(f) = \frac{2^{2n+3}((n+1)!)^4}{(2n+3)((2n+2)!)^3} f^{(2n+2)}(\xi)\,, \; \xi \in (-1,1). \tag{6.48}$$

$$I(f) - I_{GLL}(f) = -\frac{2^{2n+1}n^3(n+1)((n-1)!)^4}{(2n+1)((2n)!)^3} f^{(2n)}(\xi)\,,\ \xi \in (-1,1). \tag{6.49}$$

Due considerazioni conclusive.

1. Le formule gaussiane in $[-1,1]$ sono estendibili ad un generico intervallo $[a,b]$ con l'opportuna trasformazione lineare sia sui nodi che sui pesi.
2. In Matlab/Octave la funzione `quadl` implementa la formula di quadratura di Gauss-Lobatto. Si chiama con `quadl(fun,a,b)`: in questo caso la tolleranza di default è $1.e-3$ e `fun` può essere definita sia su un altro M-file di tipo funzione o mediante `fun=inline(' ')`. Per usare una tolleranza definita dall'utente, `tol_utente`, si userà la chiamata `quadl(fun,a,b,tol_utente)`.

Infine, facciamo un esempio di una *formula composita gaussiana* (a 2 punti). Essa generalizza infatti la formula di Gauss a 2 punti per il calcolo di

$$\int_{-1}^{1} g(t)dt \approx \sum_{i=0}^{1} A_i g(t_i)$$

con $A_i = 1,\ i = 0,1$ e $t_0 = -1/\sqrt{3}$ e $t_1 = -t_0$.

La costruzione viene fatta come segue. Partendo da una suddivisione equispaziata consideriamo, invece dei punti $x_{k-1}$ e $x_k$, i punti

$$y_{k-1} = x_{k-1} + \frac{h}{2}\left(1 - \frac{1}{\sqrt{3}}\right)\ ,\quad y_k = x_{k-1} + \frac{h}{2}\left(1 + \frac{1}{\sqrt{3}}\right)\ .$$

La formula di quadratura di Gauss composita ed il relativo errore assoluto sono:

- `Formula di Gauss composita e relativo errore.`

$$I_G^c(f) = \frac{h}{2}\sum_{k=1}^{n}\left(f(y_{k-1}) + f(y_k)\right)\ ,$$

$$I(f) - I_G^c(f) = \frac{b-a}{4320}h^4 f^{(4)}(\xi)\ ,\ \ \xi \in (a,b),$$

dove al solito $h = (b-a)/n$.

Esercizio 55. *Si calcoli numericamente*

$$\int_0^{2\pi} x e^{-x}\cos 2x dx = \frac{3(e^{-2\pi}-1) - 10\pi e^{-2\pi}}{25} \approx -0.12212260462\ ,$$

*mediante le 3 formule composite dei trapezi, di Simpson e di Gauss, per $n = 7$. Si determini anche l'errore assoluto. Se invece si prendesse $n = 10$, come cambierebbe l'approssimazione?*

Un M-file che può essere usato per implementare simultaneamente le formule composite dei trapezi, di Simpson e di Gauss dell'esercizio precedente, è descritto in Appendice C (cfr. pag. C.0.19). Si noterà che per il suo utilizzo, sarà necessario definire la funzione integranda in una funzione `funQ.m`.

## 6.3 Esercizi proposti

ESERCIZIO 56. `(Appello del 23/3/05)`. *Calcolare numericamente*

$$\int_{-1}^{1} (1+x^2)\sqrt{1-x^2}\, dx$$

*usando il metodo di Simpson composito. Quanti punti sono necessari affinchè l'errore assoluto sia* $< 1.e-4$*? Come valore esatto, considerare il valore dell'integrale ottenuto con* `quadl` *a meno di* $1.e-6$.

ESERCIZIO 57. `(Appello del 21/12/05)`. *Si calcoli un'approssimazione di*

$$I = \int_{-1}^{2} \left(\frac{5}{2}x^4 - \frac{15}{2}x^3 + 2\right) dx$$

*con le formule di Newton-Côtes di tipo chiuso con* $n \leq 4$. *Ricordiamo che le formule di Newton-Côtes di tipo chiuso hanno la forma seguente*

$$I_{n+1}(f) = \kappa \cdot h \cdot \sum_{j=0}^{n} c_j f(x_j)$$

*dove* $h = (b-a)/n$, $x_j = a+jh$, $j = 1, ..., n$ *e i coefficienti si ricavano della tabella seguente*

| $n$ | $\kappa$ | $c_0$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_5$ |
|---|---|---|---|---|---|---|---|
| 1 | 1/2 | 1 | 1 | | | | |
| 2 | 1/3 | 1 | 4 | 1 | | | |
| 3 | 3/8 | 1 | 3 | 3 | 1 | | |
| 4 | 2/45 | 7 | 32 | 12 | 32 | 7 | |
| 5 | 5/288 | 19 | 75 | 50 | 50 | 75 | 19 |

*Calcolare anche l'errore assoluto commesso rispetto al valore dell'integrale.*

ESERCIZIO 58. `(Appello del 22/6/07)`. *Un corpo in caduta libera all'equatore, subisce una deviazione dalla verticale dovuta all' accelerazione di Coriolis. Supponendo che al tempo* $t = 0$ *il corpo sia fermo (cioe' x(0)=0, v(0)=0 e a(0)=0) e che la sua accelerazione di Coriolis sia nota solo negli istanti di tempo di Tabella, si determini lo spostamento dalla verticale dovuto a tale accelerazione dopo* $t = 100$ *sec..*

*In tabella elenchiamo al variare del tempo* $t$*, i valori dell'accelerazione* $a(t)$*:*

| t | 10 | 15 | 30 | 40 | 50 | 70 | 100 |
|---|---|---|---|---|---|---|---|
| a | .0144 | .0216 | .0432 | .0576 | .072 | .1008 | .1439 |

*Mediante integrazione dell'accelerazione, il suggerimento è quindi di calcolare la velocità $v(t)$ negli istanti di tempo indicati usando la formula di quadratura dei trapezi composita e integrando nuovamente calcolare la deviazione $x(t)$ (sempre integrando numericamente con i trapezi compositi) negli stessi istanti di tempo. Essendo*

$$a = \frac{dv(t)}{dt} = \frac{d^2x(t)}{dt^2}$$

$$v(T) = \int_0^T \frac{dv(t)}{dt} dt = v(T) - v(0) \tag{6.50}$$

$$x(T) = \int_0^T \frac{dx(t)}{dt} dt = x(T) - x(0) \tag{6.51}$$

*Applicando all'equazione (6.50), la formula di quadratura composita dei trapezi, si avrebbe*

$$\begin{aligned} v(0) &= 0 \\ v(10) &= \frac{10}{2}(0.0144 + 0); \\ v(15) &= v(10) + \frac{5}{2}(0.0144 + 0.0216); \\ &\textit{ecc...} \end{aligned}$$

*Applicando ancora all'equazione (6.51), la formula di quadratura composita dei trapezi, si avrebbe*

$$\begin{aligned} x(0) &= 0 \\ x(10) &= \frac{10}{2}(v(10) + v(0)); \\ x(15) &= x(10) + \frac{5}{2}(v(10) + v(15)); \\ &\textit{ecc...} \end{aligned}$$

- *Quale sarebbe la distanza percorsa dal corpo dopo $t = 100$ sec (supponendo non ci sia attrito)? Sugg. 1 L'energia potenziale si trasforma in cinetica, quindi .... Sugg. 2 oppure per la seconda legge della dinamica*

$$m\ g = m\ \frac{d^2x}{dt^2}$$

*e integrando due volte si conclude.*

ESERCIZIO 59. *Si consideri il seguente integrale definito*

$$\int_{\frac{1}{\pi}}^{5\pi} \sin\left(\frac{1}{x}\right) dx\,.$$

1. *Dire a priori quanti punti sono necessari, sia col metodo dei trapezi composito che con il metodo di Simpson composito, per il calcolo dell'integrale a meno di tol* $= 1.e-4$. *Si suggerisce di costruire una funzione* `funQ` *che valuta sia* $f(x) = \sin(1/x)$ *che le derivate* $f^{(2)}$ *e* $f^{(4)}$.

2. *Calcolare quindi l'integrale con il metodo di Simpson composito usando il numero minimo di nodi richiesto al punto precedente. Qual è l'errore assoluto commesso? Come valore esatto usare quello ottenuto con* `quadl` *con tolleranza tol* $= 1.e-4$. *Che conclusione si può trarre osservando l'errore di approssimazione?*

3. *Calcolare l'integrale con il metodo di Simpson composito usando i punti* $x_i = (i+1)/\pi,\ i = 0, ..., 4$ *e* $x_i = (i-4)\pi,\ i = 5, \ldots, 9$. *(Sugg. Applicare Simpson composito ai due insiemi di punti sommandone poi il valore che si ottiene con Simpson nell'intervallo* $[5/\pi, \pi]$*...)*

ESERCIZIO 60. `(Appello del 11/09/07)`. *Si consideri il seguente integrale definito*

$$\int_{-\pi}^{-\frac{1}{\pi}} \sin\left(\frac{1}{x^2}\right) dx\,.$$

1. *Dire a priori, analizzando la formula dell'errore, quanti punti sono necessari per il calcolo del precedente integrale con il metodo dei trapezi composito a meno di tol* $= 1.e-3$.

2. *Calcolare quindi l'integrale con il metodo di trapezi composito usando 20 punti equispaziati tra* $-\pi$ *e* $-5/\pi$ *e 50 punti equispaziati tra* $-5/\pi$ *e* $-1/\pi$. *Qual è l'errore assoluto commesso? Usare come valore esatto quello ottenuto con la funzione* `quadl` *con la stessa tolleranza.*

ESERCIZIO 61. *Calcolare numericamente*

$$\int_{-1}^{1} \sqrt{|x^3 - 0.7|}\, dx$$

*usando il metodo dei trapezi composito su* 10 *sottointervalli di [-1,1]. Confrontare poi i risultati con la funzione* `quadl` *di Matlab usando come tolleranza* $1.e-6$.

ESERCIZIO 62. `(Appello del 29/3/07)`. *L'integrale di* $f(x) = \frac{x}{2} e^{-\frac{x}{2}} \cos(x)$ *su* $[-1, 1]$ *si può approssimare con la formula di Gauss-Legendre*

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(z_i)\ . \tag{6.52}$$

*Il vettore dei nodi* $\mathbf{z}$ *e dei pesi* $\mathbf{w}$ *si possono determinare con la M-function:*

```
 function [z,w]=zwlegendre(n)
% This function computes nodes z and weights
% w of the Gauss-Legendre quadrature formula.
%-------------------------------------------
% Input:
%      n = number of quadrature nodes
%Outputs:
%      z = column vector of the nodes
%      w = column vector of the weights
%-------------------------------------------
if n<=1
    z=[0]; w=[2];
    return
end
A=zeros(n);
k=[1:n-1];
v=k./(sqrt(4*(k.^2)-1));
A=A+diag(v,1)+diag(v,-1);
[w,z]=eig(A);
nm2=sqrt(diag(w'*w));
w=(2*w(1,:)'.^2)./nm2;
z=diag(z);
```

*Si chiede di calcolare l'integrale (6.52) con la formula di Gauss-Legendre costruita prendendo* $n = 2^i,\ i = 0, 1, ..., imax = 8$ *punti a meno di* $tol = 1.e-9$. *In pratica ci si arresterà quando* $n > 2^8$ *oppure l'errore in modulo diventa minore di* $tol$, *assumendo come valore esatto quello che si ottiene con la funzione* `quadl`*).*

## 6.4 Estrapolazione di Richardson

In questa sezione presentiamo la tecnica di *estrapolazione di Richardson* che rappresenta uno degli strumenti più interessanti per l'accelerazione di successioni, ovvero il loro calcolo "veloce", e che trova applicazione anche alla quadratura numerica.

Uno degli ingredienti su cui si basa la tecnica di Richardson è la *formula di sommazione di Eulero-Maclaurin* che a sua volta si basa sui *numeri di Bernoulli* ovvero il valore in zero dei polinomi di Bernoulli di grado pari.

Presenteremo quindi lo schema di *(estrapolazione) di Romberg* come applicazione della tecnica di Richardson alla quadratura numerica. A sua volta, la tecnica di Romberg si può pensare come l'*algoritmo di Neville* per la valutazione in 0 del polinomio di interpolazione i cui nodi non sono altro che i passi al quadrato da cui si parte per raffinare la formula di quadratura (ovvero per aumentarne l'ordine di convergenza).

Molti dei metodi numerici, quali quelli per l'interpolazione e la quadratura, si basano sulle informazioni di una certa funzione su un insieme di valori che dipende da un *passo* $h \neq 0$.

Ad ogni $h \neq 0$ posssiamo far corrispondere il valore $T(h)$ di un funzionale lineare e continuo (che rappresenta il processo numerico) che ammette un'*espansione asintotica* in termini di *potenze di h*:

$$T(h) = \tau_0 + \tau_1 h^{\gamma_1} + \tau_2 h^{\gamma_2} + \ldots + \tau_m h^{\gamma_m} + \alpha_{m+1}(h) h^{\gamma_{m+1}} \ , \ \ 0 < \gamma_1 < \gamma_2 < \cdots < \gamma_{m+1} \ , \tag{6.53}$$

con $\tau_i, \ i = 0, \ldots, m$ indipendenti da $h$, $|\alpha_{m+1}(h)| \leq A$ (ovvero limitata) e $\gamma_i$ non tutti numeri interi. Chiederemo inoltre che $\tau_0 = \lim_{h \to 0} T(h)$ ovvero, $\tau_0$ rappresenta la *soluzione* del problema considerato.

Presentiamo ora due semplici esempi di funzionali lineari che si possono rappresentare nella forma (6.53).

Esempio 44. *Sia*

$$T(h) = \frac{f(x+h) - f(x-h)}{2h}$$

*l'operatore alle* `differenze finite centrali`. *È noto che* $T(h) \approx f'(x)$. *Se* $f \in \mathcal{C}^{2m+3}[x-a, x+a]$, $m \geq 0$ *e* $|h| \leq |a|$, *allora dall'espansione di Taylor possiamo riscrivere* $T(h)$ *come segue:*

$$\begin{aligned} T(h) &= \frac{1}{2h}\left\{ f(x) + f'(x)h + f^{(2)}(x)\frac{h^2}{2!} + \ldots + \frac{h^{2m+3}}{(2m+3)!}[f^{(2m+3)}(x) + o(1)] \right\} - \\ &- \frac{1}{2h}\left\{ f(x) - f'(x)h + f^{(2)}(x)\frac{h^2}{2!} + \ldots + (-1)^{2m+3}\frac{h^{2m+3}}{(2m+3)!}[f^{(2m+3)}(x) + o(1)] \right\} = \\ &= \tau_0 + \tau_1 h^2 + \ldots + \tau_m h^{2m} + \alpha_{m+1}(h) h^{2m+2} \ , \end{aligned} \tag{6.54}$$

*dove* $\tau_0 = f'(x)$, $\tau_k = \dfrac{f^{(2k+1)}(x)}{(2k+1)!}$, $k = 1, \ldots, m+1$ *e* $\alpha_{m+1}(h) = \tau_{m+1} + o(1)$[1].

ESEMPIO 45. *Sia*

$$T(h) = \frac{f(x+h) - f(x)}{h}$$

*l'operatore alle* `differenze finite in avanti`. *Operando come prima si ha*

$$T(h) = \tau_0 + \tau_1 h + \tau_2 h^2 \ldots + \tau_m h^m + \alpha_{m+1}(h) h^{m+1} , \tag{6.55}$$

*dove* $\tau_k = \dfrac{f^{(k+1)}(x)}{(k+1)!}$, $k = 0, 1, \ldots, m+1$ *e* $\alpha_{m+1}(h) = \tau_{m+1} + o(1)$.

Infine, come già osservato alla sezione 6.1, l'operatore alle differenze finite centrali è una approssimazione migliore dell'operatore alle difference finite in avanti, poiché la sua espansione asintotica contiene solo potenze *pari* di $h$ (cfr. (6.54) e (6.55)).

La domanda d'obbligo, a questo punto, è la seguente: *come possiamo costruire un metodo generale di estrapolazione?*

Dato un *metodo di discretizzazione*, scegliamo una sequenza di passi, $\{h_0, h_1, h_2, ...\}$, tali che $h_0 > h_1 > h_2 > \ldots > 0$, e calcoliamo $T(h_i)$, $i = 0, 1, 2, ....$ Fissato poi un indice $k$, per $i \leq k$ costruiamo i *polinomi*

$$\tilde{T}_{i,k}(h) = b_0 + b_1 h^{\gamma_1} + ... + b_k h^{\gamma_k} \tag{6.56}$$

tali da soddisfare le condizioni d'*interpolazione*

$$\tilde{T}_{i,k}(h_j) = T(h_j), \quad j = i-k, i-k+1, \ldots, i \; .$$

Consideriamo quindi i valori

$$T_{i,k} = \tilde{T}_{i,k}(0)$$

come approssimazione di $\tau_0$.[2]

Ci limiteremo al caso in cui $\gamma_k = k\,\gamma$.

Poniamo, $z = h^\gamma$ e $z_j = h_j^\gamma$, $j = 0, 1, ..., m$, cosicché

$$\tilde{T}_{i,k}(h) = b_0 + b_1 z + b_2 z^2 + ... + b_k z^k := P_{i,k}(z) \; .$$

Proviamo un risultato che sostanzialmente afferma che il valore estrapolato $T_{i,k}$ altro non è che il valore in $z = 0$ del polinomio di interpolazione di grado $k$ sui nodi $z_j$, $j = i-k, ..., i$ che assume il valore $T(h_j)$.

[1] Con il simbolo $o(1)$ si intende indicare una quantità che ha ordine di infinitesimo di una costante.

[2] Talvolta ai polinomi (6.56) si preferiscono funzioni razionali.

**Proposizione 17.** *In $z = 0$,*

$$T_{i,k} := P_{i,k}(0) \overset{Lagrange}{=} \sum_{j=i-k}^{i} c_{k,j}^{(i)} P_{i,k}(z_j) = \sum_{j=i-k}^{i} c_{k,j}^{(i)} T(h_j) \tag{6.57}$$

*dove*

$$c_{k,j}^{(i)} = \prod_{\substack{s \neq j \\ s = i-k}}^{i} \frac{z_s}{z_s - z_j}$$

*sono i polinomi elementari di Lagrange, tali che*

$$\sum_{j=i-k}^{i} c_{k,j}^{(i)} z_j^p = \begin{cases} 1 & p = 0 \\ 0 & p = 1, ..., k \\ (-1)^k z_{i-k} z_{i-k+1} \cdots z_i & p = k+1 \end{cases} \tag{6.58}$$

**Dim.** Osserviamo che i coefficienti $c_{k,j}^{(i)}$ dipendono solo da $z_j$. Consideriamo i monomi $z^p,\ p = 0, 1, ..., k$. Riscriviamoli come polinomi di interpolazione di Lagrange

$$z^p = \sum_{j=i-k}^{i} z_j^p \cdot \prod_{\substack{s \neq j \\ s = i-k}}^{i} \frac{z - z_s}{z_j - z_s} \qquad p = 0, 1, ..., k\ .$$

Da cui, per $z = 0$ si ottengono le prime due uguaglianze in (6.58).

Infine, osserviamo che

$$z^{k+1} = \sum_{j=i-k}^{i} z_j^{k+1} \cdot \prod_{\substack{s \neq j \\ s = i-k}}^{i} \frac{z - z_s}{z_j - z_s} + \prod_{s=i-k}^{i} (z - z_s)\ . \tag{6.59}$$

Infatti, poiché $z^{k+1}$ sta sia a sinistra che a destra della (6.59), il polinomio differenza

$$z^{k+1} - (\textit{membro destro in } (6.59)) \in \mathbb{P}_k$$

e si annulla nei $k+1$ punti $z_s,\ s = i-k, ..., i$. Cioè esso si annulla identicamente. Ciò prova la validità della (6.59).

Sostituendo $z = 0$, si ottiene la terza delle (6.58). □

Siamo in grado di usare l'espansione (6.57). Pertanto, per $k < m$

$$T_{i,k} = \sum_{j=i-k}^{i} c_{k,j}^{(i)} T(h_j) = \sum_{j=i-k}^{i} c_{k,j}^{(i)} \left[ \tau_0 + \tau_1 z_j + \tau_2 z_j^2 + \ldots + \tau_k z_j^k + z_j^{k+1}(\tau_{k+1} + \mathcal{O}(h_j)) \right] , \tag{6.60}$$

e per $k = m$

$$T_{i,m} = \sum_{j=i-m}^{i} c_{m,j}^{(i)} T(h_j) = \sum_{j=i-m}^{i} c_{m,j}^{(i)} \left[ \tau_0 + \tau_1 z_j + \tau_2 z_j^2 + \ldots + \tau_m z_j^m + z_j^{m+1} \alpha_{m+1}(h_j) \right] . \tag{6.61}$$

Se i passi $h_j$ sono tali che $h_j = h_0 b^j,\ 0 < b < 1$, ovvero formano una successione geometrica di ragione $b$, o in generale $\frac{h_{j+1}}{h_j} \leq b < 1,\ \forall\ j$, si può dimostrare che esiste una costante $C_k$ dipendente solo da $b$ tale che

$$\sum_{j=i-k}^{i} |c_{k,j}^{(i)}| z_j^{k+1} \leq C_k z_{i-k} z_{i-k+1} \cdots z_i \ . \tag{6.62}$$

Dalle relazioni (6.58) e (6.62) segue che

$$T_{i,k} = \tau_0 + (-1)^k z_{i-k} z_{i-k+1} \cdots z_i (\tau_{k+1} + \mathcal{O}(h_{i-k})), \quad k < m \ ; \tag{6.63}$$

e

$$|T_{i,m} - \tau_0| \leq M_{m+1} C_m z_{i-m} z_{i-m+1} \cdots z_i \ , \tag{6.64}$$

se $|\alpha_{m+1}(h_j)| \leq M_{m+1}$, per $j \geq 0$.

Concludendo, per $k$ fissato e $i \to \infty$

$$|T_{i,k} - \tau_0| = \mathcal{O}(z_{i-k}^{k+1}) = \mathcal{O}(h_{i-k}^{(k+1)\gamma}) \ . \tag{6.65}$$

Rappresentando il tutto su un *"tableau"*, come in Figura 6.8, potremo dire che $T_{i,k}$, ovvero l' $i$-esimo elemento della $(k+1)$-esima colonna, converge a $\tau_0$ con ordine $(k+1)\gamma$.

### 6.4.1 Applicazione alla quadratura numerica

Sia $f \in \mathcal{C}^{2m+2}[a,b]$ e si desideri calcolare $\displaystyle\int_a^b f(t)dt$ su una *partizione uniforme*, $x_i = a + ih,\ i = 0, 1, ..., n,\ h = (b-a)/n,\ n \geq 1$.

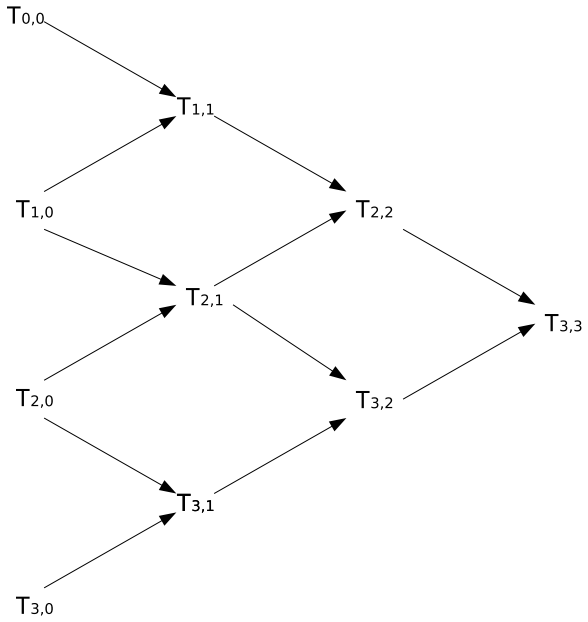


**Figura** 6.8: Tableau dello schema di Richardson per $m = 3$, con $T_{i,0} = T(h_i)$.

### Regola trapezoidale

Se si fa uso della *formula trapezoidale*, è noto che

$$T(h) = h\left(\frac{f(a)}{2} + f(a+h) + \ldots + f(b-h) + \frac{f(b)}{2}\right) .$$

Per tale funzionale vale la *formula di sommazione di Eulero-Maclaurin*

$$T(h) = \int_a^b f(t)dt + \sum_{l=1}^{m} \frac{B_{2l}h^{2l}}{(2l)!}\left(f^{(2l-1)}(b) - f^{(2l-1)}(a)\right) + h^{2m+2}\frac{B_{2m+2}}{(2m+2)!}(b-a)f^{(2m+2)}(\xi), \quad a < \xi < b . \tag{6.66}$$

La formula precedente ci da una espressione esplicita dell'errore che si commette approssimando l'integrale di $f$ su $[a, b]$ mediante la formula trapezoidale. I coefficienti $B_k$ sono i *numeri di Bernoulli* che sono definiti come il valore in 0 dei polinomi di Bernoulli di grado $k$, con $k$ pari (si veda la sottosezione 6.4.3 per alcuni cenni sui polinomi di Bernoulli).

Alla luce di quanto detto, la formula trapezoidale (6.66) si può riscrivere come

$$T(h) = \tau_0 + \tau_1 h^2 + \ldots + \tau_m h^{2m} + \alpha_{m+1}h^{2m+2} , \tag{6.67}$$

dove

$$\begin{aligned} \tau_0 &= \int_a^b f(t)dt \\ \tau_k &= \frac{B_{2k}}{(2k)!}\left(f^{(2k-1)}(b) - f^{(2k-1)}(a)\right),\ k = 1,\ldots,m \\ \alpha_{m+1}(h) &= \frac{B_{2m+2}}{(2m+2)!}(b-a)f^{(2m+2)}(\xi(h)) \quad a < \xi(h) < b\,. \end{aligned}$$

Poiché $f^{(2m+2)} \in \mathcal{C}[a,b]$, allora esiste una costante $L$ tale che $|f^{(2m+2)}(x)| \le L$, uniformemente in $x$. Ció implica che $\exists\ M_{m+1}$ tale che

$$|\alpha_{m+1}(h)| \le M_{m+1}, \quad \forall\ h = \frac{b-a}{n},\ \ n > 0\,. \tag{6.68}$$

La disequazione (6.68) ci dice che il termine di errore dell'espansione asintotica (6.67) tende a zero come $h \to 0$. Infine, detta espansione approssima $\tau_0$ come un polinomio in $h^2$, al tendere a zero di $h$.

**Metodo di Romberg**

Per il calcolo di $\tau_0$ si può procedere come segue:

1. $h_0 = b-a$, $h_1 = \dfrac{h_0}{n_1}$, $\ldots$ ,$h_m = \dfrac{h_0}{n_m}$, con $n_1,\ldots,n_m > 0$, $\ m > 0$.
2. In corrispondenza determino
$$T_{i,0} = T(h_i)\,,\ \ i = 0,1,\ldots,m\,;$$
dove $T(h)$ è il funzionale (6.67).
3. Sia
$$\tilde{T}_{m,m}(h) = a_0 + a_1h^2 + \cdots + a_mh^{2m}\,;$$
tale che $\tilde{T}_{m,m}(h_i) = T(h_i),\quad i = 0,1,\ldots,m$. Il polinomio $\tilde{T}_{m,m}$ altro non è che il polinomio di interpolazione di $T_{i,0}$.
4. Sia $\tilde{T}_{m,m}(0)$ il valore *estrapolato* di $\tau_0$.

Su queste idee si basa il *metodo di Romberg*. Le scelte dei passi $h_i$ e dei polinomi $\tilde{T}_{i,k}$ sono fatte come segue:

- $h_i = \dfrac{b-a}{2^i}$, $\ i \ge 0$.

- Per calcolare $\tilde{T}_{m,m}(0)$ (ovvero il valore estrapolato di $\tau_0$) si usa l'*algoritmo di Neville* (vedi sottosezione 6.4.4). Per $1 \le i \le k \le m$ sia $\tilde{T}_{i,k}$ il polinomio di grado $k$ in $h^2$ tale che:

$$\begin{aligned} \tilde{T}_{i,k}(h_j) &= T(h_j), \;\; j = i-k, \ldots, i \\ \tilde{T}_{i,k}(0) &= T_{i,k} \, . \end{aligned}$$

A partire da $k = 1$, l'algoritmo di Neville consente di determinare $T_{i,k}$ dai valori di $T_{i,k-1}$ e $T_{i-1,k-1}$, usando la formula

$$T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left[\frac{h_{i-k}}{h_i}\right]^2 - 1} \, , \;\; 1 \le k \le i \le m \, . \tag{6.69}$$

La formula (6.69) è l'algoritmo di Neville con $x_i = h_i^2$ (valutato in $x = 0$).

Per capire meglio il funzionamento del metodo facciamo un'esempio.

ESEMPIO 46. Calcoliamo

$$\mathcal{I} = \int_0^1 x^5 dx \, .$$

Il valore esatto dell' integrale è $\mathcal{I} = \frac{1}{6}$. Prendiamo $h_0 = 1, \; h_1 = 2^{-1}, \; h_2 = 2^{-2}$. Calcoliamo mediante la formula trapezoidale i valori $T_{0,0} = 0.5$ corrispondente a $h_0^2$, $T_{1,0} = 0.265625 \approx \frac{17}{64}$ corrispondente a $h_1^2$ e $T_{2,0} = 0.192383 \approx \frac{197}{1024}$ corrispondente a $h_2^2$. Usiamo la (6.69) per calcolare $T_{1,1}$ e $T_{2,1}$. Un ulteriore applicazione della (6.69) consente di determinare $T_{2,2} = 0.1666667 \approx \frac{1}{6}$.

Una prima importante proprietà dello schema di Romberg è che ogni $T_{i,k}$ del tableau costruito con la (6.69) (vedi Figura 6.8) rappresenta una *regola di integrazione lineare*, ovvero

$$T_{i,k} = \alpha_0 f(a) + \alpha_1 f(a + h_i) + \cdots + \alpha_{n_i - 1} f(b - h_i) + \alpha_{n_i} f(b) \, .$$

**Proposizione 18.** *Per $i = k$ alcune formule $T_{k,k}$ rappresentano formule di quadratura di tipo Newton-Cotes. In particolare*

- *$T_{0,0}$ è la formula dei trapezi ($T_{i,0}$ formule dei trapezi composte);*
- *$T_{1,1}$ è la formula di Simpson, ($T_{i,1}$ formule di Simpson composte);*
- *$T_{2,2}$ è la formula di Milne.*

*$T_{3,3}$ non è una formula di Newton-Cotes.*

**Dim.** Facilmente si prova che

$$T_{0,0} = \frac{b-a}{2}(f(a)+f(b)) \ , \ (formula\ dei\ trapezi)$$

$$T_{1,0} = \frac{b-a}{2^2}(f(a)+2f(\frac{a+b}{2})+f(b)) \ .$$

Da cui, mediante l'algoritmo di Neville

$$T_{1,1} = T_{1,0} + \frac{T_{1,0}-T_{0,0}}{3} \ = \ \frac{4}{3}T_{1,0} - \frac{1}{3}T_{0,0} \ .$$

Sviluppando

$$T_{1,1} = \frac{b-a}{2}\left(\frac{1}{3}f(a) + \frac{4}{3}f\left(\frac{a+b}{2}\right) + \frac{1}{3}f(b)\right) \ ,$$

che è la ben nota formula di Simpson.

Le rimanenti affermazioni si lasciano per esercizio. □

Come ultima osservazione, il metodo di Romberg è un metodo di estrapolazione di Richardson della formula (6.53) in cui l'esponente $\gamma_k = 2k$.

### 6.4.2 Una implementazione del metodo di Romberg

Il metodo di Romberg per la quadratura si applica usando la seguente *ricetta*: si costruisce una tabella, **T** triangolare (inferiore), la cui prima colonna consiste dei valori approssimati dell'integrale mediante formule composite dei trapezi costruite usando suddivisioni regolari con $N = 2^m, \ m = 0, 1, 2, ....,$ (ovvero suddivisioni con $2^m + 1$ punti). Se indichiamo con $T_{i,1}, \ i = 1, 2, \ldots$ l' elemento dell' $i$-esima riga della prima colonna di **T**, che contiene il valore approssimato dell'integrale con i passi $h_i = 2^{-i}$, ovvero $2^i + 1$ punti, gli elementi delle successive colonne sono costruiti mediante la ricorrenza

$$T_{i,k} = \frac{4^k T_{i,k-1} - T_{i-1,k-1}}{4^k - 1} \ , \ i = k, \ldots, m \ , \ k = 0, \ldots, m \ , . \tag{6.70}$$

Un esempio di tabella di Romberg è visualizzato in Tabella 6.5. Questa tecnica trova la sua utilità nelle seguenti due proprietà

(a) $T_{N,k}$ è una formula di quadratura del tipo

$$T_{N,k} = \sum_{j=1}^{N} A_{j,N} f(x_{j,N}) \ .$$

| | | | | |
|---|---|---|---|---|
| $T_{2^0,0}$ | | | | |
| $T_{2,0}$ | $T_{2^0,1}$ | | | |
| $T_{2^2,0}$ | $T_{2^1,1}$ | $T_{2^0,2}$ | | |
| $T_{2^3,0}$ | $T_{2^2,1}$ | $T_{2^1,2}$ | $T_{2^0,3}$ | |
| $T_{2^4,0}$ | $T_{2^3,1}$ | $T_{2^2,2}$ | $T_{2,3}$ | $T_{1,4}$ |
| $\vdots$ | $\vdots$ | | $\ddots$ | |

Tabella 6.5: Tabella del metodo di Romberg

(b) Ciascuna delle formule in una data riga, come ad esempio la riga evidenziata in Tabella 6.5

$$T_{2^3,0},\ T_{2^2,1},\ T_{2^1,2},\ T_{2^0,3}$$

o in generale

$$T_{2^m,0},\ T_{2^{m-1},1},\ T_{2^{m-2},2},\ T_{2^{m-3},3}, ....(*)$$

è una formula con $N = 2^m + 1$ punti e in ciascuna delle formule (*) i punti sono gli stessi che in $T_{2^m,0}$.

Infine, vale il seguente risultato.

**Teorema 25.** *Ciascuna formula $T_{1,k}, T_{2,k}, T_{3,k}, ....$ è una formula di grado di esattezza $2k - 1$.*

Ad esempio, se consideriamo la terza colonna di Tabella 6.5, essa rappresenta una formula di quadratura esatta sui polinomi di grado 5 (ecco perchè integra perfettamente la funzione $x^5$).

### 6.4.3 I polinomi di Bernoulli

In questa sottosezione desideriamo richiamare alcune delle caratteristiche salienti dei polinomi di Bernoulli.

Si parte dall'intervallo $I = [0, 1]$ e per ogni $x \in I$ i polinomi di Bernoulli sono definiti

dalle seguenti relazioni:

$$B_0(x) = 1\,, \tag{6.71}$$

$$B_1(x) = x - \frac{1}{2}\,, \tag{6.72}$$

$$B'_{k+1}(x) = (k+1)B_k(x), \quad k = 1, 2, .... \tag{6.73}$$

Le relazioni (6.72) e (6.73) consentono di determinare i polinomi di Bernoulli a meno di una costante di integrazione. Per avere univocità si introducono le ulteriori condizioni

$$B_{2l+1}(0) = 0 = B_{2l+1}(1), \quad l \geq 1\,. \tag{6.74}$$

Si voglia ad esempio determinare $B_2(x)$. Dalle (6.72) e (6.73) si avrebbe $B'_2(x) = 2x - 1$. Integrando $B_2(x) = x^2 - x + c$. Usando ancora le (6.72) e (6.73) si avrebbe $B_3(x) = x^3 - \frac{3}{2}x^2 + 3cx + d$. Usando le *condizioni al contorno* (6.74) si ottiene $d = 0,\ c = \frac{1}{6}$.

Da quanto detto segue che i numeri di Bernoulli sono *nulli* per i polinomi di grado dispari (ciò segue da (6.74)) e diversi da zero per quello di grado pari. I primi 4 numeri pari di Bernoulli sono: $B_0 = 1,\ B_2 = \frac{1}{6},\ B_4 = -\frac{1}{30},\ B_6 = \frac{1}{42}$.

Due proprietà facilmente verificabili sono:

1. $(-1)^k B_k(1-x) = B_k(x), \quad k \geq 0$;
2. $\int_0^1 B_k(t)dt = 0, \quad k > 1$.

Per il grafico di alcuni polinomi di Bernoulli, vedasi Fig. 6.9.

### 6.4.4 Algoritmo di Neville

L'algoritmo di Neville consente di valutare il polinomio interpolante mediante una successione di interpolazioni lineari di polinomi di grado via via crescente.

Sia $S_n = \{(x_i, y_i),\ i = 0, 1, ..., n\}$ un insieme di punti in $\mathbb{R}^2$ Nella sua forma originale l'algoritmo funziona come segue:

(a) Fase di inizializzazione

$$P_{i,0} = y_i, \quad i = 0, 1, ..., n\,.$$

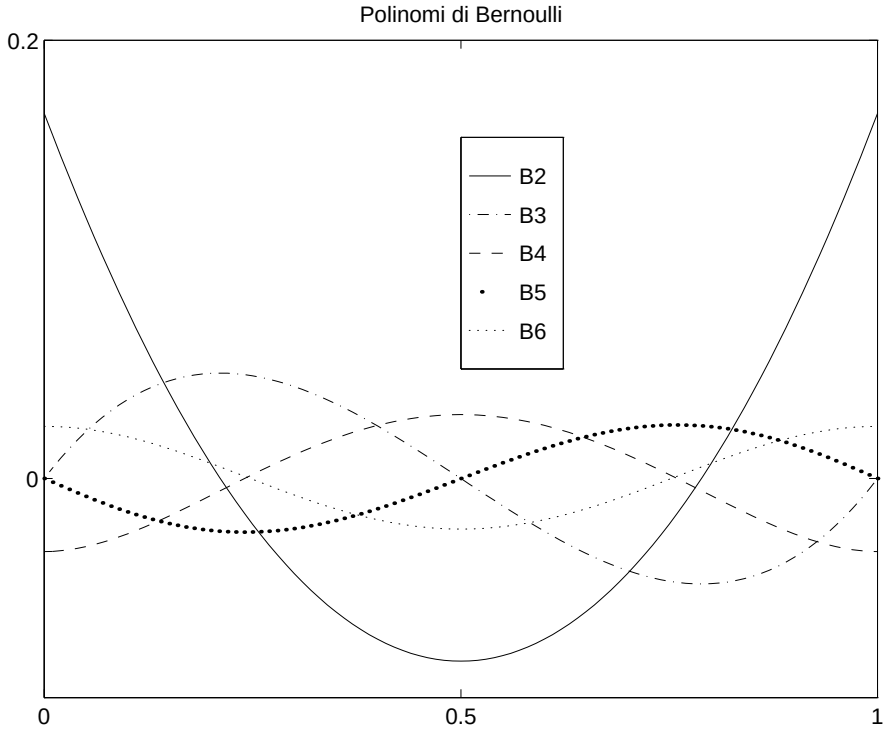


**Figura** 6.9: Alcuni polinomi di Bernoulli.

(b) Passo iterativo

$$P_{i,k} = \frac{x - x_{i-k}}{x_i - x_{i-k}} P_{i,k-1} + \frac{x_i - x}{x_i - x_{i-k}} P_{i-1,k-1}\,,$$

$$P_{i,k} = P_{i,k-1} + \frac{P_{i,k-1} - P_{i-1,k-1}}{\frac{x-x_{i-k}}{x-x_i} - 1}\,, \quad i = 1, ..., n \;\; k = i, ..., n\,.$$

Al termine del processo $P_{n,n}$ conterrà il valore in $x$ del polinomio di interpolazione di grado $n$ su $S_n$.

```
function [n]=neville(x,y,t)
% ----------------------------------------------------
% Valuta in t il polinomio di interpolazione di
% grado length(x)-1, mediante l'algoritmo di Neville
% facendo uso di un solo vettore p
%-----------------------------------------------------
n=length(x); p=y;
for i=2:n,
```

```
        for k=i:n,
            p(k)=(p(k)*(t-x(k-i+1))-p(k-1)*(t-x(k)))/(x(k)-x(k-i+1));
        end
end
n=p(n);
return
```

Il polinomio interpolante ottenuto con lo schema di Neville, può scriversi nella forma

$$P_{i,k}(x) = \sum_{j=i}^{i+k} l_{j,i}^{k}(x) y_j$$

dove i polinomi di grado $k$, $l_{j,i}^{k}(x)$, sono i polinomi elementari di Lagrange.

Tale algoritmo si può applicare allo schema di Romberg pur di prendere $x = 0$ e $x_i = h_i^2$ nonché prendendo $i = 0, 1, 2, ...$ e $k = 1, ..., i$ nel passo iterativo.

# Appendice A

# Metodi iterativi ed equazione logistica

## A.1 Malthus e Verhlust

Iniziamo questa prima appendice ricordando due tra i più noti e semplici modelli di evoluzione di una popolazione.

### A.1.1 Modello lineare di Malthus

Il Rev.do Thomas (Robert) Malthus (?/2/1766- 23/12/1834), curato inglese ad Albury (vicino ad Oxford), nel suo saggio *"An Essay on the Principle of Population"* pubblicato nel 1798, ipotizzò che una popolazione che non ha scambi con l'esterno cresce sempre più dei propri mezzi di sussistenza.

**Figura** A.1: Thomas Malthus

Aveva delle visioni pessimistiche sia come demografo che come economista. Predisse che la crescita di una popolazione matematicamente è una crescita geometrica, ovvero il tasso di crescita è lineare.

Se pertanto $x_0$ è il numero di individui iniziali, allora dopo un certo tempo la popolazione sarà $x_1 = x_0 + g\,x_0$, con $g \in \mathbb{R}$ che è detto `fattore di crescita` (o `growth rate`). Allora $x_1 = (1+g)x_0$, $x_2 = (1+g)x_1 = (1+g)[(1+g)x_0] = (1+g)^2 x_0$

e al passo $k$

$$x_k = (1+g)^k x_0 \quad g \in \mathbb{R} \tag{A.1}$$

che è una progressione geometrica di ragione $1+g$.

*Domanda*: come varia la popolazione? *Risposta*: in funzione di $g$ e del valore iniziale $x_0$.

Studiamo la successione (o progressione) geometrica (A.1). Essa converge **se e solo se** $|1+g| < 1$ per ogni popolazione iniziale $x_0$. Pertanto, si ha **convergenza** quando $-2 < g < 0$. Se $-2 < g \leq -1$ allora $-1 < 1+g < 0$ cosicché $x_k$ sarà negativo per $k$ dispari e positivo altrimenti. Ovvero non sapremo dire nulla. Se $g = -1$, $1+g = 0$ e quindi $x_k = 0, \quad \forall k$. Infine, quando $-1 < g < 0$, $1+g < 1$ per cui $x_k < x_0$: la *popolazione si estingue!*

Ci sono due altri casi da considerare:

- $g = 0$. In tal caso la popolazione rimane inalterata $x_k = x_0, \quad \forall k$.
- **Divergenza** quando $g > 0$. Infatti, se $1+g > 1$ che implica $x_k > x_{k-1} > \cdots > x_0$: *la popolazione cresce esponenzialmente.*

Esempio 47. *Come esempio, consideriamo la popolazione iniziale $x_0 = 100$ e consideriamo 10 iterazioni, $k = 0, 1, ..., 10$. L'evoluzione sarà come in Figura A.2*

### A.1.2 Il modello non lineare di Verhulst

Pierre Verhulst (Brussels, 28/10/1804-15/2/1849) era un matematico che si interessò di biologia e in particolare della legge di crescita di una popolazione.

Nel 1838 in Verhulst, P. F. *Notice sur la loi que la population pursuit dans son accroissement*, Corresp. Math. Phys. 10:113-121, propose un nuovo modello di crescita della popolazione, assumendo non più una crescita costante ma con fattore di crescita di tipo lineare $g(x) = -ax + b, \quad a > 0$. Partendo da una popolazione iniziale $x_0$, la (A.1) al passo $k$ si scriverà come

$$x_{k+1} = x_k + g(x_k)\,x_k = -ax_k^2 + (1+b)x_k\,. \tag{A.2}$$

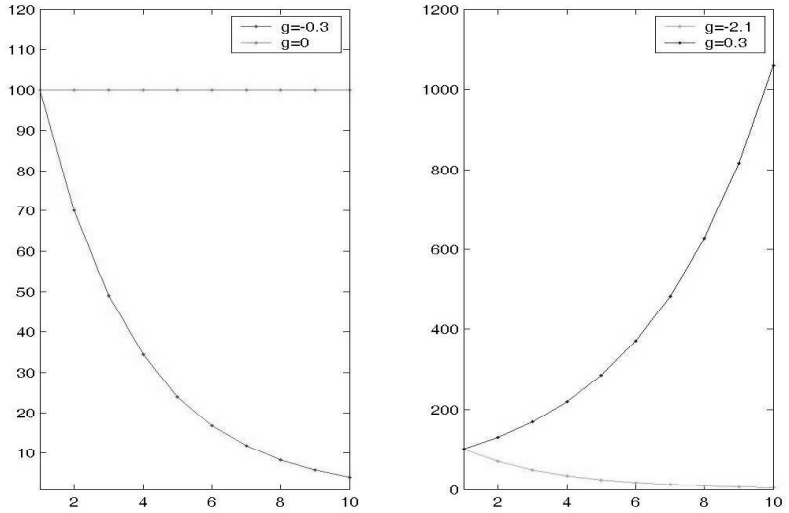


**Figura** A.2: La progressione di Malthus a partire da una popolazione iniziale di 100 individui per diversi valori di $g$.

L'equazione (A.2) ha senso se $a > -1$ e $0 \leq x \leq \frac{1+b}{a}$ (perchè la popolazione deve essere sempre $\geq 0$). Il modello è equivalente alla mappa quadratica

$$\begin{aligned} &T : \mathbb{R}^+ \longrightarrow \mathbb{R}^+ \\ &x \to T(x) = -ax^2 + (1+b)x \ . \end{aligned} \tag{A.3}$$

Consideriamo la trasformazione lineare $x = \frac{(1+b)}{a}y$, che mappa l'intervallo $[0, (1+b)/a]$, dove la parabola di (A.3) è $T(x) \geq 0$ in $[0, 1]$. Otteniamo

$$\tilde{T}(y) = -a\left(\frac{1+b}{a}\right)^2 y^2 + (1+b)\cdot\left(\frac{1+b}{a}\right)y \tag{A.4}$$

Semplificando

$$\tilde{T}(y) = -\kappa y^2 + \kappa y \tag{A.5}$$

avendo posto $\kappa = \frac{(1+b)^2}{a}$, vedi Fig. A.4.

Possiamo allora studiare la mappa discreta

$$x_{k+1} = -\kappa\, x_k^2 + \kappa\, x_k \ , \ \ 0 < \kappa \leq 4. \tag{A.6}$$

Il processo iterativo (A.6) si chiama `processo logistico discreto`. Pertanto, partendo da un $x_0 \in (0, 1]$, scelto un valore di $\kappa \in (0, 4]$, itereremo la mappa (A.6) un certo numero di volte, ottenendo un punto del cosidetto **diagramma di Verhulst**.

Riassumendo, indichiamo in tabella A.1, le differenze tra i due approcci.

**Figura** A.3: Pierre Verhlust

| | *Malthus*: **lineare** | *Verhulst*: **non lineare** |
|---|---|---|
| fattore di crescita | costante $g$ | lineare $g(x) = ax + b$ |
| processo | $\begin{cases} x_0 & start \\ x_{n+1} = (1+g)x_n \end{cases}$ | $\begin{cases} x_0 & start \\ x_{n+1} = -kx_n^2 + kx_n & k = (1+b)^2/a \end{cases}$ |
| trasformazione | $T(x) = (1+g)x$ | $T(x) = -kx^2 + kx$ |

Tabella A.1: Tabella di confronto tra le iterazioni di Malthus e Verhlust

### A.1.3 Isometrie, dilatazioni e contrazioni

In entrambi i procedimenti di Malthus e Verhulst, partendo da un $x_0$ e da una funzione $T : \mathbb{R} \to \mathbb{R}$ si è generata una successione di valori $\{x_n\}_{n\geq 0}$ tale che $x_{n+1} = T(x_n),\ n = 0, 1, ....$ Consideriamo allora la successione

$$x_{n+1} = T(x_n),\ n = 0, 1, 2, ... \tag{A.7}$$

essa sarà detta

- una **isometria** se

$$|T(x_{n+1}) - T(x_n)| = |x_{n+1} - x_n|, \quad \forall n \in \mathbb{N} \tag{A.8}$$

- una **dilatazione** se

$$|T(x_{n+1}) - T(x_n)| > |x_{n+1} - x_n|, \quad \forall n \in \mathbb{N} \tag{A.9}$$

- oppure una **contrazione** se

$$|T(x_{n+1}) - T(x_n)| < \kappa\,|x_{n+1} - x_n|, \quad \forall n \in \mathbb{N},\ \ \kappa \in [0, 1) \tag{A.10}$$

Vale il seguente *Teorema del punto fisso di (Banach-)Caccioppoli*

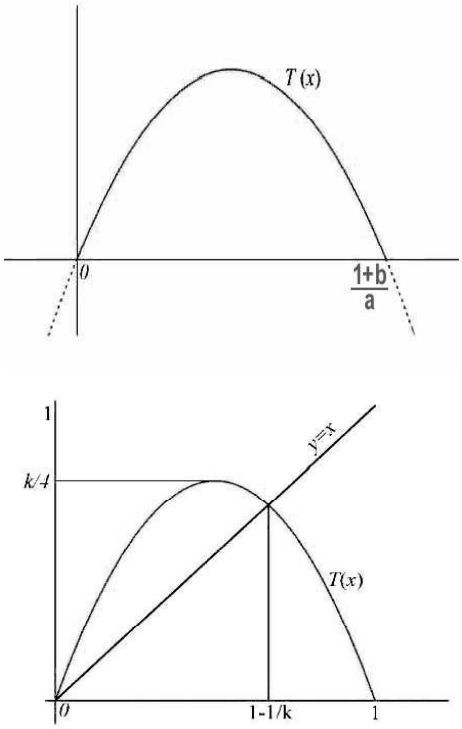


**Figura** A.4: La trasformazione lineare della parabola $T(x) \geq 0$ in $[0,1]$

**Teorema 26.** *Ogni* `contrazione` $T : \mathbb{R} \to \mathbb{R}$ *ammette un* **unico** *punto fisso,*

$$x^\star = T(x^\star).$$

*Partendo da un fissato* $x_0$ *il processo*

$$x_{n+1} = T(x_n)\,, \quad n = 0, 1, 2, ...$$

*è un'approssimazione di* $x^\star$ *che migliora ad ogni passo, cioè*

$$|x_{n+1} - x^\star| < |x_n - x^\star|, \quad n = 0, 1, 2, ...$$

Il punto $x^\star$ è detto appunto **punto fisso della contrazione** $T$.

Se $T$ è una contrazione, allora esiste un $\kappa \in [0, 1)$ tale che

$$|T(x_{n+1}) - T(x_n)| < \kappa\, |x_{n+1} - x_n|, \quad \forall n \in \mathbb{N}, \tag{A.11}$$

ovvero

$$\frac{|T(x_{n+1}) - T(x_n)|}{|x_{n+1} - x_n|} < 1\,. \tag{A.12}$$

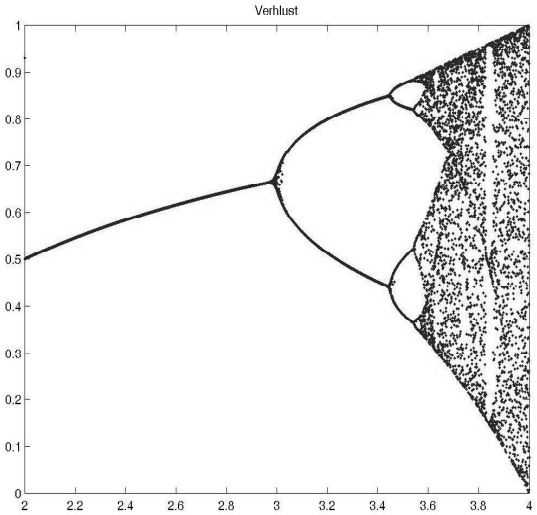


**Figura** A.5: Iterazione del processo di Verhulst che origina il ben noto *diagramma di biforcazione*

La disuguaglianza (A.12) ci dice che il "rapporto incrementale" è sempre minore di 1. Quando $|x_{n+1} - x_n| \leq \epsilon$ allora il rapporto incrementale approssima la **derivata** di $T$ in $x_n$.

### A.1.4 Esempi di processi iterativi

1. **Processo di traslazione**: $T(x) = x + a$. Le progressioni aritmetiche, come la capitalizzazione semplice degli interessi, sono processi di traslazione. Sono processi isometrici.

2. **Processo omotetico**: $T(x) = mx$.

   - $|m| < 1$, è una contrazione.
   - $|m| > 1$,è una dilatazione.
   - $|m| = 1$, è una isometria (identità se $m = 1$ e simmetria rispetto l'origine se $m = -1$).

   Le progressioni geometriche, quali la capitalizzazione composta degli interessi, sono processi omotetici.

**Rappresentazione grafica di processi iterativi**

**Figura** A.6: Renato Caccioppoli

1. In un riferimento cartesiano ortogonale, tracciamo il grafico della trasformazione $T$ (linea blu) e della funzione identica $y = x$ (linea verde).

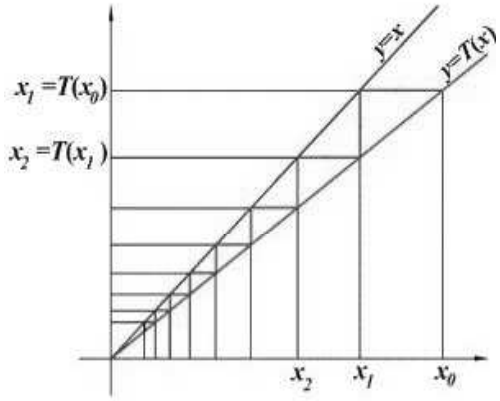


**Figura** A.7: Rappresentazione di un processo iterativo.

2. Fissato un punto iniziale $x_0$, costruiamo la sua immagine $x_1 = T(x_0)$ sull'asse delle ordinate. Per simulare il procedimento di `retroazione`, riportiamo in ascissa il valore $x_1$ attraverso la funzione identica $y = x$. Costruiamo l'immagine $x_2 = T(x_1)$ del punto $x_1$ e riportiamo il suo valore in ascissa e procediamo iterativamente. Nel primo fotogramma della figura seguente è illustrato un processo shift, nel secondo e nel terzo due processi omotetici di contrazione con attrattore nullo. Il quarto fotogramma rappresenta infine un processo espansivo.

Esempio 48. Esempi di un processo iterativo convergente Fig. A.9 e di processo iterativo divergente Fig. A.10. Infine alcune iterazioni di Verhulst per diversi valori di $\kappa$ si trovano nelle figure Fig. A.11-A.13.

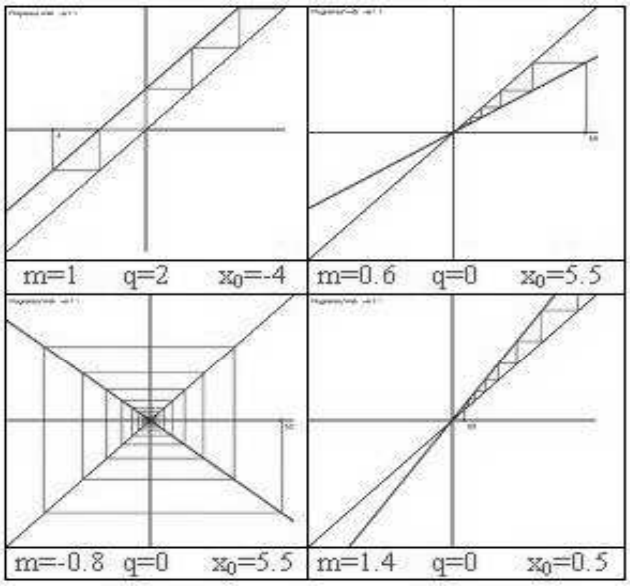


**Figura** A.8: Processi iterativi per diversi valori di $m$

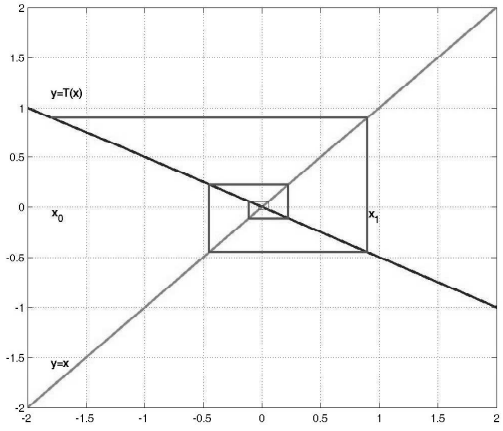


**Figura** A.9: Processo convergente

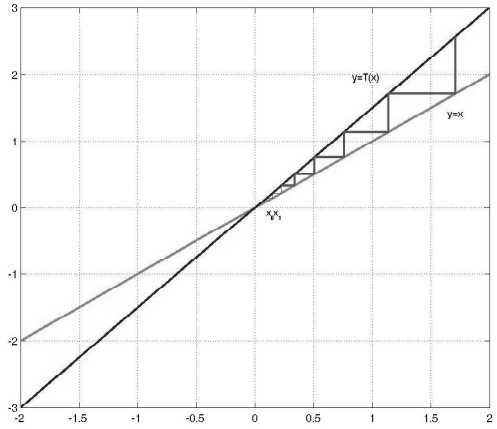


**Figura** A.10: Processo divergente

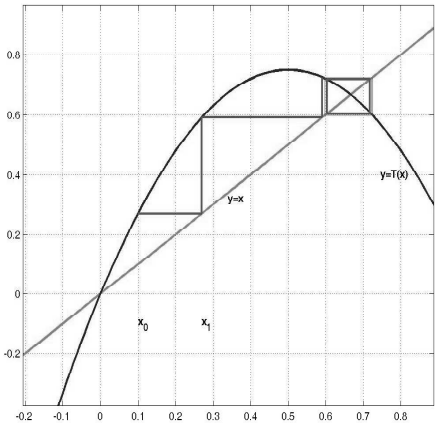


**Figura** A.11: Processo di Verhulst convergente con $x_0 = 0.1, \kappa = 3$.

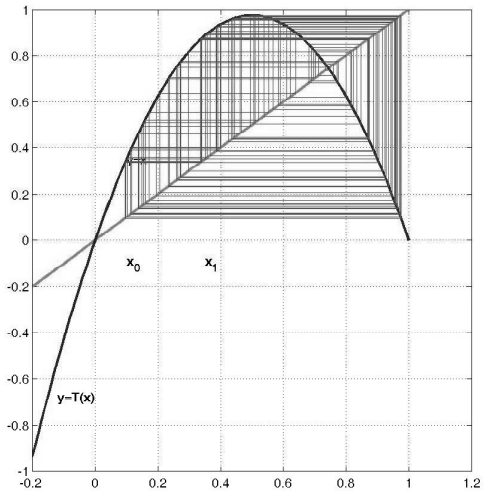


**Figura** A.12: Processo di Verhulst convergente con $x_0 = 0.1, \kappa = 3.9$

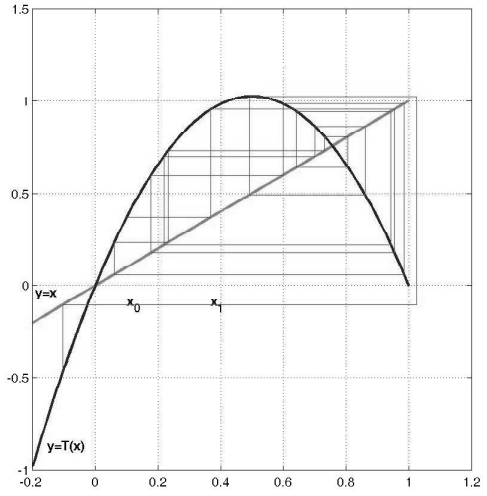


**Figura** A.13: Processo di Verhulst divergente con $x_0 = 0.1, \kappa = 4.1$

# Appendice B

# Aspetti implementativi dell'interpolazione polinomiale

## B.1 Richiami sull'interpolazione polinomiale

Data una funzione $f\colon [a,b] \to \mathbb{R}$ e un insieme $\{x_i\}_{i=1}^n \subset [a,b]$, sia $p_{n-1}f(x)$ il polinomio di grado $n-1$ interpolatore di $f$ nei punti $x_i$ (cioè $p_{n-1}f(x_i) = f(x_i)$. Chiameremo i punti $x_i$ *nodi di interpolazione* (o, più semplicemente, *nodi*). Un generico punto $\bar{x} \in [a,b]$ in cui si valuta $L_{n-1}f$ sarà chiamato *nodo target* (o, più semplicemente, *target*).

I $n$ *nodi di Chebyshev* sono gli zeri del polinomio di Chebyshev di grado $n$ $T_n(x) = \cos(n \arccos(x))$. Dunque, $x_{j+1} = \cos\left(\frac{j\pi+\frac{\pi}{2}}{n}\right)$, $j = 0, \ldots, n-1$. Si chiamano $n$ *nodi di Chebyshev estesi (o di Chebyshev–Lobatto)* i nodi $\bar{x}_{j+1} = \cos\left(\frac{j\pi}{n-1}\right)$, $j = 0, \ldots, n-1$. Tali nodi appartengono all'intervallo $[-1,1]$. I nodi di Chebyshev relativi ad un intervallo generico $[a,b]$ si ottengono semplicemente per traslazione e scalatura.

### B.1.1 Interpolazione di Lagrange

Dato un insieme di $n$ coppie di interpolazione $\{(x_i, y_i)\}_{i=1}^n$, il polinomio elementare di Lagrange $i$-esimo (di grado $n-1$) è

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{x_i - x_j} \, .$$

L'algoritmo per il calcolo dei polinomi di Lagrange su vettori (colonna) target $x$ è riportato in Tabella B.1.

```
function y = lagrange(i,x,nodi)
%
% y = lagrange(i,x,nodi)
%
n = length(nodi);
m = length(x);
y = prod(repmat(x,1,n-1)-repmat(nodi([1:i-1,i+1:n]),m,1),2)/...
prod(nodi(i)-nodi([1:i-1,i+1:n]));
```

Tabella B.1: Polinomio elementare di Lagrange.

Il polinomio di interpolazione si scrive dunque

$$p_{n-1}(x) = \sum_{i=1}^{n} y_i L_i(x) \ .$$

### B.1.2 Sistema di Vandermonde

Dato il polinomio

$$p_{n-1}(x) = a_1 x^{n-1} + a_2 x^{n-2} + \ldots + a_{n-1}x + a_n$$

e $n$ coppie di interpolazione $\{(x_i, y_i)\}_{i=1}^n$, il corrispondente sistema di Vandermonde si scrive

$$\begin{pmatrix} x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \ldots & x_2 & 1 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \ldots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{pmatrix} \tag{B.1}$$

L'implementazione dell'algoritmo per il calcolodella matrice di Vandermonde è riportata in Tabella B.2. Alternativamente, si può usare la function di Matlab/Octave `vander`.

```
function V = vandermonde(nodi)
%
% V = vandermonde(nodi)
%
n = length(nodi);
V = repmat(nodi',1,n).^repmat([n-1:-1:0],n,1);
```

Tabella B.2: Matrice di Vandermonde.

### B.1.3 Interpolazione di Newton

Data una funzione $f$, definiamo le differenze divise nel seguente modo:

$$\begin{aligned} f[x] &= f(x) \\ f[x_1, x] &= \frac{f[x] - f[x_1]}{x - x_1} \\ \ldots &= \ldots \\ f[x_1, x_2, \ldots, x_{k-1}, x_k, x] &= \frac{f[x_1, x_2, \ldots, x_{k-1}, x] - f[x_1, x_2, \ldots, x_{k-1}, x_k]}{x - x_k} \end{aligned}$$

L'algoritmo per il calcolo delle differenze divise è riportato in Tabella B.3.

```
function d = diffdiv(nodi,valori)
%
% d = diffdiv(nodi,valori)
%
n = length(nodi);
for i = 1:n
  d(i) = valori(i);
  for j = 1:i-1
    d(i) = (d(i)-d(j))/(nodi(i)-nodi(j));
  end
end
```

Tabella B.3: Differenze divise.

L'interpolazione nella forma di Newton si scrive dunque

$$\begin{aligned} p_0 f(x) &= d_1 \\ w &= (x - x_1) \\ p_i f(x) &= p_{i-1} f(x) + d_{i+1} w, \quad i = 1, \ldots, n-1 \\ w &= w \cdot (x - x_{i+1}), \qquad i = 1, \ldots, n-1 \end{aligned}$$

ove

$$d_i = f[x_1, \ldots, x_i] \, .$$

Il calcolo delle differenze divise e la costruzione del polinomio di interpolazione possono essere fatti nel medesimo ciclo `for`.

Sfruttando la rappresentazione dell'errore

$$f(x) - p_{i-1} f(x) = \left( \prod_{j=1}^{i} (x - x_j) \right) f[x_1, \ldots, x_i, x] \approx \left( \prod_{j=1}^{i} (x - x_j) \right) f[x_1, \ldots, x_i, x_{i+1}] \tag{B.2}$$

è possibile implementare un algoritmo per la formula di interpolazione di Newton adattativo, che si interrompa cioè non appena la stima dell'errore è più piccola di una fissata tolleranza.

Dato il polinomio interpolatore nella forma di Newton

$$p_{n-1}(x) = d_1 + d_2(x - x_1) + \ldots + d_n(x - x_1) \cdot \ldots \cdot (x - x_{n-1}) \,,$$

si vede che le differenze divise soddisfano il sistema lineare

$$\begin{pmatrix} 0 & \ldots & \ldots & 0 & 1 \\ 0 & \ldots & 0 & (x_2 - x_1) & 1 \\ \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \prod_{j=1}^{n-2}(x_{n-1} - x_j) & \ldots & (x_{n-1} - x_1) & 1 \\ \prod_{j=1}^{n-1}(x_n - x_j) & \ldots & \ldots & (x_n - x_1) & 1 \end{pmatrix} \begin{pmatrix} d_n \\ d_{n-1} \\ \vdots \\ d_2 \\ d_1 \end{pmatrix} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{pmatrix} \tag{B.3}$$

### B.1.4 Interpolazione polinomiale a tratti

Data una funzione $f\colon [a,b] \to \mathbb{R}$ e un'insieme $\{x_i\}_{i=1}^n \subset [a,b]$ di nodi ordinati, consideriamo l'interpolante polinomiale a tratti $L_{k-1}^c f$ di grado $k-1$. Su ogni intervallo $h_i = x_{i+1} - x_i$ essa coincide con il polinomio di grado $k-1$

$$a_{i,1}(x - x_i)^{k-1} + a_{i,2}(x - x_i)^{k-2} + \ldots + a_{i,k-1}(x - x_i) + a_{i,k} \,. \tag{B.4}$$

Dunque, l'interpolante polinomiale a tratti è completamente nota una volta noti i nodi e i coefficienti di ogni polinomio.

### B.1.5 Strutture in Matlab/Octave

In Matlab/Octave è possibile definire delle *strutture*, cioè degli insiemi (non ordinati) di oggetti. Per esempio, le istruzioni

```
S.a = 1;
S.b = [1,2];
```

generano la struttura `S`

```
S =
{
  a =  1
  b =
```

```
    1   2

}
```

L'interpolazione polinomiale a tratti è definita mediante una struttura solitamente chiamata `pp` *(piecewise polynomial)*, che contiene gli oggetti `pp.x` (vettore colonna dei nodi), `pp.P` (matrice dei coefficienti), `pp.n` (numero di polinomi), `pp.k` (grado polinomiale aumentato di uno) e `pp.d` (numero di valori assunti dai polinomi). La matrice $P$ ha dimensione $n \times k$ e, con riferimento a (B.4),

$$P_{ij} = a_{i,j} \, .$$

Nota una struttura `pp`, è possibile valutare il valore dell'interpolante in un generico target $\bar{x}$ con il comando `ppval(pp,xbar)`.

### B.1.6 Splines cubiche

Le splines cubiche sono implementate da Matlab/Octave con il comando `spline` che accetta in input il vettore dei nodi e il vettore dei valori e restituisce la struttura associata. La spline cubica costruita è nota come *not-a-knot*, ossia viene imposta la continuità della derivata terza (generalemente discontinua) nei nodi $x_2$ e $x_{n-1}$. Lo stesso comando permette di generare anche le splines *vincolate*: è sufficiente che il vettore dei valori abbia due elementi in più rispetto al vettore dei nodi. Il primo e l'ultimo valore verranno usati per imporre il valore della derivata alle estremità dell'intervallo.

#### Implementazione di splines cubiche naturali in Matlab/Octave

Con le notazioni usate fino ad ora, si può costruire una spline cubica $S$ a partire dalla sua derivata seconda nell'intervallo generico $[x_i, x_{i+1}]$

$$S''_{[x_i,x_{i+1}]}(x) = \frac{m_{i+1} - m_i}{h_i}(x - x_i) + m_i, \quad i = 1, \ldots, n-1 \tag{B.5}$$

ove $m_i = S''(x_i)$ sono incogniti, con $m_1 = m_n = 0$. Integrando due volte la (B.5), si ottiene

$$\begin{aligned} S'_{[x_i,x_{i+1}]}(x) &= \frac{m_{i+1} - m_i}{2h_i}(x - x_i)^2 + m_i(x - x_i) + a_i \\ S_{[x_i,x_{i+1}]}(x) &= \frac{m_{i+1} - m_i}{6h_i}(x - x_i)^3 + \frac{m_i}{2}(x - x_i)^2 + a_i(x - x_i) + b_i \end{aligned}$$

ove le costanti $a_i$ e $b_i$ sono da determinare. Innanzitutto, richiedendo la proprietà di interpolazione, cioè $S_{[x_i,x_{i+1}]}(x_j) = f(x_j)$, $j = i, i+1$, si ottiene

$$\begin{aligned} b_i &= f(x_i), \\ a_i &= \frac{f(x_{i+1}) - f(x_i)}{h_i} - (m_{i+1} - m_i)\frac{h_i}{6} - m_i\frac{h_i}{2} = \\ &= \frac{f(x_{i+1}) - f(x_i)}{h_i} - m_{i+1}\frac{h_i}{6} - m_i\frac{h_i}{3} \end{aligned}$$

A questo punto, richiedendo la continuità della derivata prima, cioè $S'_{[x_{i-1},x_i]}(x_i) = S'_{[x_i,x_{i+1}]}(x_i)$ per $i = 2, \ldots, n-1$, si ottiene

$$\frac{h_{i-1}}{6}m_{i-1} + \frac{h_{i-1}+h_i}{3}m_i + \frac{h_i}{6}m_{i+1} = \frac{f(x_{i+1}) - f(x_i)}{h_i} - \frac{f(x_i) - f(x_{i-1})}{h_{i-1}}\,. \tag{B.6}$$

Risulta chiaro che ci sono $n-2$ equazioni e $n$ incognite $m_i$.

**Splines cubiche *naturali*** Si impone che il valore della derivata seconda agli estremi dell'intervallo sia 0. Dunque $m_1 = m_n = 0$. Il sistema lineare (B.6) diventa allora

$$\begin{pmatrix} 1 & 0 & \ldots & \ldots & \ldots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \ldots & \ldots & \ldots & 0 & 1 \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

con $d_1 = d_n = 0$ e $d_i = \frac{f(x_{i+1})-f(x_i)}{h_i} - \frac{f(x_i)-f(x_{i-1})}{h_{i-1}}$, $i = 2, \ldots, n-1$. L'algoritmo per il calcolo della struttura associata ad una spline cubica naturale è riportato in Tabella B.4.

**Splines cubiche *vincolate*** Si impongono due valori $d'_1$ e $d'_2$ per la derivata $S'(x_1)$ e $S'(x_n)$, rispettivamente. Si ricava dunque

$$\begin{aligned} a_1 &= d'_1 \\ \frac{m_n - m_{n-1}}{2h_{n-1}}(x_n - x_{n-1})^2 + m_{n-1}(x_n - x_{n-1}) + a_{n-1} &= d'_n \end{aligned}$$

da cui

$$\begin{aligned} \frac{h_1}{3}m_1 + \frac{h_1}{6}m_2 &= \frac{f(x_2) - f(x_1)}{h_1} - d'_1 \\ \frac{h_{n-1}}{6}m_{n-1} + \frac{h_{n-1}}{3}m_n &= d'_n - \frac{f(x_n) - f(x_{n-1})}{h_{n-1}} \end{aligned}$$

```
function pp = splinenaturale(x,y)
%
% function pp = splinenaturale(x,y)
%
n = length(x);
x = x(:);
y = y(:);
h = x(2:n)-x(1:n-1);
d1 = h(2:n-2)/6;
d0 = (h(1:n-2)+h(2:n-1))/3;
rhs = (y(3:n)-y(2:n-1))./h(2:n-1)-(y(2:n-1)-y(1:n-2))./h(1:n-2);
S = diag(d1,-1)+diag(d0)+diag(d1,1);
m = zeros(n,1);
m(2:n-1) = S\rhs;
a = (y(2:n)-y(1:n-1))./h(1:n-1)-h(1:n-1).*(m(2:n)/6+m(1:n-1)/3);
b = y(1:n-1);
pp.x = x;
pp.P = [(m(2:n)-m(1:n-1))./(6*h),m(1:n-1)/2,a,b];
pp.k = 4;
pp.n = n-1;
pp.d = 1;
```

Tabella B.4: Spline cubica naturale.

Il sistema lineare da risolvere diventa dunque

$$\begin{pmatrix} \frac{h_1}{3} & \frac{h_1}{6} & 0 & \dots & \dots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \dots & \dots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

con $d_1 = \frac{f(x_2)-f(x_1)}{h_1} - d_1'$ e $d_n = d_n' - \frac{f(x_n)-f(x_{n-1}}{h_{n-1}}$.

**Splines cubiche *periodiche*** Si impone $S''(x_1) = S''(x_n)$ e $S'(x_1) = S'(x_n)$. Si ricava dunque

$$\begin{aligned} m_1 &= m_n \\ a_1 &= \frac{m_n - m_{n-1}}{2} h_{n-1} + m_{n-1} h_{n-1} + a_{n-1} \end{aligned}$$

da cui

$$m_1 - m_n = 0$$
$$\frac{h_1}{3}m_1 + \frac{h_1}{6}m_2 + \frac{h_{n-1}}{6}m_{n-1} + \frac{h_{n-1}}{3}m_n = \frac{f(x_2)-f(x_1)}{h_1} - \frac{f(x_n)-f(x_{n-1})}{h_{n-1}}$$

Il sistema lineare da risolvere diventa dunque

$$\begin{pmatrix} 1 & 0 & \dots & \dots & \dots & 0 & -1 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & \dots & 0 \\ 0 & \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ \frac{h_1}{3} & \frac{h_1}{6} & 0 & \dots & 0 & \frac{h_{n-1}}{6} & \frac{h_{n-1}}{3} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

con $d_1 = 0$ e $d_n = \frac{f(x_2)-f(x_1)}{h_1} - \frac{f(x_n)-f(x_{n-1})}{h_{n-1}}$.

**Splines cubiche *not-a-knot*** Si impone la continuità della derivata terza in $x_2$ e $x_{n-1}$. Si ricava dunque

$$\frac{m_2 - m_1}{h_1} = \frac{m_3 - m_2}{h_2}$$
$$\frac{m_{n-1} - m_{n-2}}{h_{n-2}} = \frac{m_n - m_{n-1}}{h_{n-1}}$$

da cui

$$\frac{1}{h_1}m_1 - \left(\frac{1}{h_1} + \frac{1}{h_2}\right)m_2 + \frac{1}{h_2}m_3 = 0$$
$$\frac{1}{h_{n-2}}m_{n-2} - \left(\frac{1}{h_{n-2}} + \frac{1}{h_{n-1}}\right)m_{n-1} + \frac{1}{h_{n-1}}m_n = 0$$

Il sistema lineare da risolvere diventa dunque

$$\begin{pmatrix} \frac{1}{h_1} & -\frac{1}{h_1} - \frac{1}{h_2} & -\frac{1}{h_2} & 0 & \dots & 0 \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} & \frac{h_{n-1}}{6} \\ 0 & \dots & 0 & \frac{1}{h_{n-2}} & -\frac{1}{h_{n-2}} - \frac{1}{h_{n-1}} & \frac{1}{h_{n-1}} \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ \vdots \\ \vdots \\ m_{n-1} \\ m_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{pmatrix}$$

con $d_1 = d_n = 0$.

**Rappresentazione dell'errore**

Supponiamo di usare un metodo di interpolazione polinomiale a tratti di grado $k-1$ in un intervallo $[a,b]$ e consideriamo due diverse discretizzazioni, rispettivamente con $n_1$ e $n_2$ nodi, con intervalli di lunghezza media $h_1 = (b-a)/(n_1-1)$ e $h_2 = (b-a)/(n_2-1)$. Gli errori di approssimazione saranno verosimilmente $\text{err}_1 = Ch_1^k$ e $\text{err}_2 = Ch_2^k$. Si ha dunque

$$\frac{\text{err}_2}{\text{err}_1} = \left(\frac{h_2}{h_1}\right)^k$$

da cui

$$\log \text{err}_2 - \log \text{err}_1 = k(\log h_2 - \log h_1) = -k(\log(n_2-1) - \log(n_1-1)) \ .$$

Dunque, rappresentando in un grafico logaritmico-logaritmico l'errore in dipendenza dal numero di nodi, la pendenza della retta corrisponde al grado di approssimazione del metodo, cambiato di segno.

### B.1.7 Compressione di dati

Supponiamo di avere un insieme molto grande di coppie di nodi/valori $\{(x_i, y_i)\}_{i=1}^N$ e di non conoscere la funzione che associa il valore al nodo corrispondente. Ci poniamo il problema di *comprimere* i dati, ossia memorizzare il minor numero di coefficienti pur mantenendo un sufficiente grado di accuratezza. Una prima idea potrebbe essere quella di selezionare alcuni dei nodi, diciamo $n$, e di costruire la spline cubica su quei nodi. Il costo di memorizzazione, oltre ai nodi, sarebbe dunque pari a $4(n-1)$. Rimarrebbe il problema di scegliere i nodi da memorizzare, visto che non si suppone siano equispaziati.

Si potrebbe ridurre il costo di memorizzazione (a $n$) usando un unico polinomio interpolatore: rimarrebbe il problema della scelta dei nodi e, probabilmente, si aggiungerebbe un problema di mal condizionamento sempre dovuto alla scelta dei nodi.

Un'idea che combina le tecniche discusse è la seguente: si usa una interpolazione a tratti (anche lineare) per ricostruire i valori della funzione sconosciuta in corrispondenza di $n$ nodi di Chebyshev. Si usa poi un unico polinomio interpolatore su quei nodi. Il rapporto di compressione è $2N/n$, considerando che non è necessario memorizzare i nodi di Chebyshev, ma solo i coefficienti del polinomio interpolatore (e trascurando i due estremi dell'intervallo).

### B.1.8 Esercizi proposti

ESERCIZIO 63. *Si implementi una function* `y = lagrange(i,x,nodi)` *che valuta il polinomio di Lagrange i-esimo nel vettore* $x$.

ESERCIZIO 64. *Si implementi una function* `y = interplagrange(nodi,valori,x)` *per la formula di interpolazione nella forma di Lagrange.*

ESERCIZIO 65. *Si testi l'interpolazione nella forma di Lagrange della funzione di Runge nell'intervallo* $[-5,5]$ *su nodi equispaziati. Si prendano rispettivamente* $n = 11, 21, 31, 41, 51$ *nodi di interpolazione e si valuti l'interpolante su* $5(n-1)+1$ *nodi target equispaziati. Si producano delle figure mettendo in evidenza i nodi di interpolazione, la funzione di Runge e l'interpolante.*

ESERCIZIO 66. *Si implementi una function* `y = chebyshev(n)` *per il calcolo dei nodi di Chebyshev nell'intervallo* $[-1,1]$.

ESERCIZIO 67. *Si ripeta l'esercizio 65 usando nodi di interpolazione di Chebyshev anziché nodi equispaziati.*

ESERCIZIO 68. *Si implementi una function* `V = vandermonde(nodi)` *per il calcolo della matrice di Vandermonde definita in* (B.1).

ESERCIZIO 69. *Si implementi una function* `y = interpvandermonde(nodi,valori,x)` *per la formula di interpolazione mediante matrice di Vandermonde. Si spieghino i risultati ottenuti.*

ESERCIZIO 70. *Si ripeta l'esercizio 65, usando la formula di interpolazione mediante matrice di Vandermonde. Si usi il metodo di Hörner (vedasi Sezione 2.6.1, tabella 2.2) per la valutazione del polinomio.*

ESERCIZIO 71. *Si implementi una function* `y = interpnewton(nodi,valori,x)` *per il calcolo del polinomio di interpolazione nella forma di Newton.*

ESERCIZIO 72. *Si ripeta l'esercizio 65, usando la formula di interpolazione di Newton.*

ESERCIZIO 73. *Si modifichi l'implementazione dell'interpolazione nella forma di Newton, in modo da prevedere come parametro opzionale di input la tolleranza per l'errore (in norma infinito) di interpolazione, stimato come in* (B.2). *Nel caso la tolleranza non sia raggiunta, l'algoritmo si interrompe all'ultimo nodo di interpolazione. La function deve fornire in uscita il numero di iterazioni e la stima dell'errore.*

ESERCIZIO 74. *Si considerino* $n = 21$ *nodi di interpolazione equispaziati nell'intervallo* $[-5,5]$. *Si interpoli in forma di Newton la funzione* $y = \cos(x)$ *sull'insieme di nodi target* $\{-2,0,1\}$ *per diverse tolleranze e, successivamente, sull'insieme di nodi target* $\{-2\pi, \pi\}$. *Si spieghino i risultati ottenuti.*

ESERCIZIO 75. *Si calcolino i numeri di condizionamento della matrice di Vandermonde* (B.1) *e della matrice dei coefficienti dell'interpolazione di Newton, da ordine 2 a 20 (considerando nodi equispaziati in* $[-1,1]$ *e se ne produca un grafico semilogaritmico nelle ordinate. Si discutano i risultati.*

ESERCIZIO 76. *Si implementi una function* `pp = lintrat(x,y)` *per l'interpolazione lineare a tratti.*

ESERCIZIO 77. *Si verifichi, mediante un grafico logaritmico-logaritmico, il grado di approssimazione (errore in norma infinito) delle splines cubiche naturali per la funzione di Runge. Si considerino un numero di nodi di interpolazione equispaziati nell'intervallo* $[-5, 5]$ *da* $n = 11$ *a* $n = 91$ *e 102 nodi target equispaziati.*

ESERCIZIO 78. *Si ripeta l'esercizio precedente con l'interpolazione lineare a tratti.*

ESERCIZIO 79. *Data la struttura associata ad una spline cubica, si ricavi la corrispondente struttura per la derivata seconda.*

ESERCIZIO 80. *Si ripeta l'esercizio 77, confrontando però la derivata seconda della funzione di Runge e la derivata seconda della spline cubica not-a-knot associata.*

ESERCIZIO 81. *Si considerino le coppie* $\{(x_i, y_i)\}$ *ove gli* $x_i$ *sono* $N = 1001$ *nodi equispaziati nell'intervallo* $[0, 2\pi]$ *e* $y_i = \sin(x_i)$. *Mediante il procedimento descritto in § B.1.7 (interpolazione lineare a tratti e interpolazione su nodi di Chebyshev estesi), si determini il minimo grado n necessario per comprimere i dati con un errore in norma infinito inferiore a* $10^{-5}$. *Si determini poi l'errore in corrispondenza del rapporto di compressione 286. Infine, si giustifichi la stagnazione dell'errore di approssimazione per grado di interpolazione maggiore di 10.*

# Appendice C

# Codici Matlab/Octave

### C.0.9 Bisezione

```
function [sol,k]=bisezione(a,b,tol)
%-----------------------------------
% Metodo di bisezione
% La funzione fun.m descrive la
% funzione di cui si cerca la radice
%-----------------------------------
% Inputs
%  a,b : estremi dell'intervallo
%  tol : tolleranza
% Output
%  sol : la soluzione cercata
%  k   : numero d'iterazioni fatte
%---------------------------------
if fun(a)*fun(b) > 0
    error('L'' intervallo non contiene alcuna radice');
elseif
    abs(fun(a)*fun(b) ) < tol
    error('Uno degli estremi e'' gia'' sulla radice ')
else
    a0=a; b0=b; k=0;
    disp('Numero iterazioni da fare a priori : ')
    ceil((log(abs(b0-a0))-log(tol))/log(2))

while abs(b-a)>tol*abs(b),
    m=(a+b)/2;
    if abs(fun(m))< tol
      disp('La radice cercata e' : ') m
      break;
```

```
    elseif fun(m)*fun(a)<0,
        b=m;
    else
        a=m;
    end
  k=k+1;
end
end
disp('La radice cercata e' : '); sol=(a+b)/2

disp('Numero iterazioni effettuate : '); k
return
```

### C.0.10 Metodo di iterazione funzionale

```
function [x0, niter, flag]=MetIterazioneFunz(x0, tol, kmax)

x1=g(x0); k=1; flag=1;
while abs(x1-x0) > tol*abs(x1) & k <= kmax
  x0=x1;
  x1=g(x0);
  k=k+1;
end
% Se converge, x0 oppure x1 contengono il valore
% dello zero cercato. Altrimenti, si pone flag=0
% che indica la non convergenza
if (k > kmax)
  flag=0;
end
niter=k-1;
return
```

### C.0.11 Metodo di Newton e molteplicità delle radici

```
function[x,iter,stimaerr,m]=NewtonRadiciMult(f,fder,x0,tol,maxit,varargin)
% ------------------------------------------------
% Function che determina la radice di f e
% la molteplicita' con il metodo di Newton
% ------------------------------------------------
% inputs  f: funzione; fder: derivata di f
%         x0: valore iniziale
% output  x: radice; iter: iterazioni eseguite
```

```
%         stimaerr: stima errore; m: molteplicita'
%
%-------------------------------------------
m0 = 1; x = x0; iter = 1;

stimaerr= -feval(f,x,varargin{:})/feval(fder,x,varargin{:});

x = x+stimaerr; x1 = x; iter = iter+1;

stimaerr = -feval(f,x,varargin{:})/feval(fder,x,varargin{:});

m = m0+1;

while (abs(stimaerr) > tol) & (iter < maxit) & (abs(m-m0)> m/100)
  m0 = m;
  x = x+stimaerr;
  iter = iter+1;
  stimaerr = -feval(f,x,varargin{:})/feval(fder,x,varargin{:});
  m = (x1-x0)/(2*x1-x-x0);
  x0 = x1;
  x1 = x;
end

stimaerr = stimaerr*m;

while (abs(stimaerr) > tol) & (iter < maxit)
  x = x+stimaerr;
  iter = iter+1;
  stimaerr = -m*feval(f,x,varargin{:})/feval(fder,x,varargin{:});
end
 stimaerr = abs(stimaerr);
  if (stimaerr > tol)
    warning('Impossibile raggiungere la tolleranza richiesta')
    warning('entro il numero massimo di iterazioni consentito.')
  end
```

### C.0.12 Metodo di accelerazione di Aitken

```
function [alfa]=Aitken(g,x0,tol,kmax)
% g e' la funzione di iterazione g(x)=x-f(x)
k=0; x1=g(x0); x2=g(x1);
xnew=x0-(x1-x0)^2/(x2-2*x1+x0);
 while abs(x2-xnew)>tol & k <=kmax)
    x0=xnew;
    x1=g(x0);
    x2=g(x1);
```

```
    xnew=x0-(x1-x0)^2/(x2-2*x1+x0);
    k=k+1;
 end
% Al termine, alfa sara' l'ultimo valore di xnew
alfa=xnew;
return
```

### C.0.13 Soluzione di sistemi lineari con metodi iterativi

```
function [xn,i,flag]=jacobi(a,f,xv,nmax,toll)
%----------------------------------------
% Metodo iterativo di Jacobi per sistemi lineari.
%----------------------------------------
% Parametri in ingresso:
%  a :     Matrice del sistema
%  f :     Termine noto (vettore riga)
%  xv :    Vettore iniziale (vettore riga)
%  nmax :  Numero massimo di iterazioni
%  toll :  Tolleranza sul test d'arresto (fra iterate)
%
% Parametri in uscita
%  xn :    Vettore soluzione
%  i :     Iterazioni effettuate
%  flag:   se flag=1 converge altrimenti flag=0
%-------------------------------------------------------
flag=1; i=1; d=diag(diag(a));
J=-inv(d)*(a-d); q=inv(d)*f;
xn=J*xv+q;

while (i<=nmax & norm(xn-xv)>toll)
   xv=xn;
   xn=J*xv+q;
   i=i+1;
end
if i>nmax,
  disp('** Jacobi non converge nel numero di iterazioni fissato');
  flag=0;
end return
```

```
function [sol,iter,err]=GaussSeidel(A,b,x0,tol,kmax)
% ---------------------------------------------
% Funzione che implementa il metodo iterativo
% di Gauss-Seidel
```

```
%-----------------------------------------------
% Inputs
% A, b: matrice e termine noto, rispettivamente
% x0  : guess iniziale
% tol : tolleranza calcoli
%
% Outputs
% sol : vettore soluzione
% iter: numero delle iterazioni
%-----------------------------------------------
n=length(x0);

D=diag(diag(A)); L=tril(A)-D; U=triu(A)-D; DI=inv(D+L); GS=-DI*U;
disp('raggio spettrale matrice iterazione di Gauss-Seidel');
max(abs(eig(GS)))
b1=DI*b;
x1=GS*x0+b1;
k=1;
err(k)=norm(x1-x0,inf);

while(err(k)>tol*norm(x0,inf) & k<=kmax)
    x0=x1;
    x1=GS*x0+b1;
    k=k+1;
    err(k)=norm(x1-x0,inf);
end

sol=x1; iter=k-1;
```

```
function [xv,iter,flag]=GaussRil(a,f,xin,nmax,toll,omega)
%-----------------------------------------------------------
% Metodo di Gauss-Seidel rilassato per sistemi lineari.
% Per omega uguale a 1 si ottiene il metodo di Gauss Seidel
% altrimenti e' il metodo SOR
%-----------------------------------------------------------
% Parametri di ingresso:
%  a :     Matrice del sistema
%  f :     Termine noto (vettore riga)
%  xin :   Vettore iniziale (vettore riga)
%  nmax :  Numero massimo di iterazioni
%  toll :  Tolleranza sul test d'arresto (fra iterate)
%  omega : Parametro di rilassamento
%
% Parametri in uscita
%  xv :    Vettore soluzione
%  iter :  Iterazioni effettuate
```

```
%  flag :  se flag=1 converge altrimenti flag=0
%-------------------------------------------------------------

flag=1; [n,m]=size(a); d=diag(a); dm1=ones(n,1)./d; dm1=diag(dm1);
b=eye(size(a))-dm1*a; g=dm1*f; bu=triu(b); bl=tril(b);
%-------------
% Iterazioni
%------------
xv=xin; xn=xv;
i=0;
while i<nmax,
  for j=1:n;
    xn(j)=(1-omega)*xv(j)+omega*(bu(j,:)*xv+bl(j,:)*xn+g(j));
  end;
  if abs(xn-xv)<toll,
     iter=i;
     i=nmax+1;
  else
     dif=[dif;norm(xn-xv)];
     xv=xn;
     i=i+1;
  end,
end if i==nmax,
   disp('** Non converge nel numero di iterazioni fissato')
   flag=0;
end
```

```
function [omega0]=SOROmegaZero(d,b,n)
%-------------------------------------------
% Inputs
%  d : vettore elementi diagonale principale
%  b : vettore elementi extradiagonali
%  n : lunghezza vettore d
% Output
%  omega0 : il parametro ottimale w0
%-------------------------------------------
% costruisco la matrice
A=diag(d)-diag(b,1)-diag(b,-1);

d1=diag(d); b1=tril(A)-d1; c1=triu(A)-d1;

w=linspace(.1,1.9,100); for i=1:100
 p(i)=max(abs(eig(inv(d1-w(i)*b1)*((1-w(i))*d1+w(i)*c1))));
end;

plot(w,p);
```

```
xlabel(' \omega '); ylabel(' p(H_\omega) ');
title('Raggio spettrale della matrice del metodo SOR');
[mp,imp]=min(p);
omega0=w(imp);
```

### C.0.14 Autovalori di matrici

#### Localizzazione di autovalori di matrici

```
function CerchiGerschgorin(A)
%-------------------------------------------------
% Costruiamo i cerchi di Gerschgorin di una matrice A
%-------------------------------------------------
tol=1.e-10; Amod=abs(A); n=max(size(A));
raggi=sum(Amod,2)-diag(Amod); xc=real(diag(A)); yc=imag(diag(A));

% angoli per il disegno dei cerchi
th=[0:pi/100:2*pi]; x=[]; y=[];
 figure(1);
 clf;
 axis equal;
 hold on;

for i=1:n,
    x=[x; raggi(i)*cos(th)+xc(i)];
    y=[y; raggi(i)*sin(th)+yc(i)];
    patch(x(i,:),y(i,:),'red');
end
% disegno il bordo e il centro dei cerchi
for i=1:n,
    plot(x(i,:),y(i,:),'k',xc(i),yc(i),'ok');
end

xmax=max(max(x)); ymax=max(max(y)); xmin=min(min(x));
ymin=min(min(y)); hold off; figure(2);
 clf;
 axis equal;
 hold on;
%-------------------------------------------------------------------
% I cerchi lungo le colonne... sono quelli della matrice trasposta
%-------------------------------------------------------------------
raggi=sum(Amod)-(diag(Amod))'; x=[]; y=[]; for i=1:n,
    x=[x; raggi(i)*cos(th)+xc(i)];
    y=[y; raggi(i)*sin(th)+yc(i)];
    patch(x(i,:),y(i,:),'green');
end
```

```
% disegno il bordo e il centro dei cerchi
for i=1:n,
    plot(x(i,:),y(i,:),'k',xc(i),yc(i),'ok');
end
%Determino il bounding box per il plot ottimale
xmax=max(max(max(x)),xmax); ymax=max(max(max(y)),ymax);
xmin=min(min(min(x)),xmin); ymin=min(min(min(y)),ymin); hold off;
axis([xmin xmax ymin ymax]);
figure(1); axis([xmin xmax ymin ymax]);
return
```

### Metodo delle potenze per il calcolo dell'autovalore di modulo massimo

```
function [lam,x,iter]=MetPotenze(A,tol,kmax,x0)
%-----------------------------------------------
% Inputs
% A: matrice,
% tol: tolleranza;
% kmax: numero massimo d'iterazioni
% x0: vettore iniziale
%--
% Outputs
% lam : autovalore di modulo massimo
% x: autvettore corrispondente
% iter: iterazioni effettuate
%-----------------------------------------------
x0=x0/norm(x0); lam=x0'*(A*x0); err=tol*abs(lam)+1; iter=0;

while (err > tol*abs(lam) & abs(lam)~=0 & iter<=kmax)
    x=A*x0;
    x=x/norm(x);
    lamnew=x'*(A*x);
    err=abs(lamnew-lam);
    lam=lamnew;
    x0=x;
    iter=iter+1;
end
return
```

### Metodo di deflazione o di Bernoulli per il calcolo degli autovalori di una matrice

```
function [rad]=metBernoulli(c,nmax,tol)
%-----------------------------------------------------
```

```
% inputs:
%     c: vettore dei coefficienti del polinomio
%  nmax: numero max iterazioni per il metodo delle potenze
%   tol: tolleranza richiesta
% output:
%   rad: vettore delle radici richieste
%-------------------------------------------------------

c1=c(1:length(c)-1)/c(length(c)); %normalizzo
n=length(c1);

% Costruisco la matrice di Frobenius

F(n,:)=-c1; F=F+diag(ones(1,n-1),1);
%-----------------------------------------------
% Applico la funzione MetPotenze per il calcolo
% dell'autovalore di modulo max della matrice F
% a partire da un vettore t0 formato da tutti 1
%-----------------------------------------------
t0=ones(n,1);
[lam1,t1,iter]=MetPotenze(F,tol,namx,t0);
rad(1)=lam1;
%-------------------------------------------------------------
% Calcolo del secondo autovalore-radice per deflazione
% Costruisco la matrice di Householder P t.c. P*t1=(1,0,...,0)
%-------------------------------------------------------------
t12=norm(t1,2);
beta=1/(t12*(t12+abs(t1(1)) ));
v(1)=sign(t1(1))*(abs(t1(1))+t12);
v(2:n)=t1(2:n);
P=eye(n)-beta*v'*v;

F1=P*F*P';

%-------------------------------------------------------------
% Calcolo i rimanenti autovalori per deflazione
% y, la prima volta, indichera' l'autovettore corrispondente
% all'autovalore lam(1).
%-------------------------------------------------------------
for s = 2:n
  m=length(y);
  t12=norm(y,2);
  beta=1/(t12*(t12+abs(y(1))));
  v(1)=sign(y(1))*(abs(y(1))+t12);
  v(2:m)=y(2:m);
  P=eye(m)-beta*v'*v;

  F1=P*F*P';
```

```
  F=F1(2:end,2:end);
  x=rand(m,1);
  [rad(s), y, iter] = MetPotenze(F,tol,nmax,x);
  clear v
end
disp('Controllo usando la funzione roots')
norm(roots(c(end:-1:1))-lam')
```

### Metodo QR per a ricerca di tutti gli autovalori di una matrice

```
function [T,iter]=MetQR(A,tol,kmax)
%-------------------------------------------------------
% Metodo QR per il calcolo di tutti gli autovalori di una
% matrice data.
%-------------------------------------------------------

[Q,R]=qr(A); T=Q*R;
%.................................................
% QQ e' la matrice contenente il prodotto comulativo
% delle matrici ortogonali Q_k, ovvero
% QQ=\prod_{k=1}^M Q_k.
%.................................................
QQ=Q; k=1; while convergenzaQR(T,tol)==0 & k <= kmax,
    T1=R*Q;
    [Q,R]=qr(T1);
    QQ=QQ*Q;
    T=Q*R;
    k=k+1;
end
disp('Numero iterazioni ' ) k
% Verifica

disp('Calcolo del residuo diag(T)-eig(A) ' )
norm(diag(T),2)-norm(eig(A),2) iter=k-1;



function nn=convergenzaQR(T,tol)
%-----------------------------------------------------------
% Controlla che gli elementi extradiagonali della matrice
% T sono sotto la tolleranza
%-----------------------------------------------------------
[n1,n2]=size(T); if n1 ~= n2,
    error('Attenzione .... matrice non quadrata');
    return
```

```
end if sum(abs(diag(T,-1))) <= tol,
    nn=1;
else
    nn=0;
end return
```

### Metodo QR con shift

```
function [T,iter]=MetQRShift(A,tol,kmax)

n=size(A,1);
T=hess(A); % forma di Hessenberg di A
iter=1;

for k=n:-1:2,
    I=eye(k);
    while convergenzaQRShift(T,tol,k)==0 & iter <= kmax,
     mu=T(k,k);
     [Q,R]=qr(T(1:k,1:k)-mu*I);
     T(1:k,1:k)=R*Q+mu*I;
     iter=iter+1;
    end
T(k,k-1)=0;
end
```

dove il test di convergenza, si farà implementando una funzione `convergenzaQRShift` per la disuguaglianza (4.9).

### Ricerca di tutti gli autovalori di una matrice simmetrica

```
function [D,iter,phi]=symJacobi(A,tol)
%-----------------------------------------------------------------------
% Data la matrice simmetrica A e una tolleranza tol, determina,
% mediante il metodo di Jacobi tutti i suoi autovalori che memorizza
% nella matrice diagonale D. Determina inoltre il numero di iterazioni,
% iter, e la quantita' phi che contiene la norma degli elementi
% extra-diagonali e come converge il metodo.
%
% Richiede tre funzioni
% calcoloCeS         : calcola coseno e seno
% calcoloProdottoGtDG: determina il prodotto G'DG
% phiNorm            : \sqrt(\sum_{i=1:n-1,j=i+1:n}^n a_{ij}^2+a_{ji}^2)
%-----------------------------------------------------------------------
```

```
n=max(size(A)); D=A; phiD=norm(A,'fro'); epsi=tol*phiD;
phiD=phiNorm(D); iter=0; [phi]=phiD;

while (phiD > epsi)
    iter=iter+1;
    for p=1:n-1,
        for q=p+1:n
            [c,s]=calcoloCeS(D,p,q);
            D=calcoloProdottoGtDG(D,c,s,p,q);
        end;
    end
    phiD=phiNorm(D);
    phi=[phi; phiD];
end
return
```

### C.0.15 Interpolazione polinomiale

#### Forma di Lagrange

```
function l=lagrai(z,x,i)
%-------------------------------------------------
% Calcola l'i-esimo pol. elementare di Lagrange
% z = nodi di interpolazione
% x = punto su cui valutare
% i = indice del polinomio
%-------------------------------------------------
z1=setdiff(z,[z(i)]);
l=prod(x-z1)/prod(z(i)-z1);
return

\bigskip

function l = lagrai_target(z,x,i)
%-------------------------------------------------
% Calcola l'i-esimo pol. elementare di Lagrange
% z = nodi di interpolazione
% x = vettore (colonna!) di punti target
%     su cui valutare l_i
% i = indice del polinomio
%
% l = vettore dei valori di l_i sui targets
%-------------------------------------------------
n = length(z); m = length(x);
```

```
l = prod(repmat(x,1,n-1)-repmat(z([1:i-1,i+1:n]),m,1),2)/...
prod(z(i)-z([1:i-1,i+1:n])); return
```

```
function leb=CostLebesgue(x)
%-------------------------------------------------------------
% Dato il vettore x di N nodi (ordinato), la funzione calcola
% il valore della costante di Lebesgue per i=2,...,N
%
% Input: x (vettore ordinato dei nodi)
% Output: leb (vettore dei valori della costante di Lebesgue)
%-------------------------------------------------------------
a=x(1); b=x(end); M=1000;
xx=linspace(a,b,M); %sono i punti "target"
N=length(x);

for i=2:N,
 for s=1:i
     l(s,:)=lagrai_target(x,xx,s);
 end
leb(i-1)=norm(l,1);
end
return
```

### Differenze divise

```
function [b]=DiffDivise(x,y)
%-------------------------------------------------
% Algoritmo delle differenze divise
%-------------------------------------------------
% Inputs
% x: vettore dei punti di interpolazione,
% y: vettore dei valori della funzione.
% Output
% b: vettore delle differenze divise b=[b_1,....b_n]
%-------------------------------------------------
n=length(x); b=y; for i=2:n,
    for j=2:i,
       b(i)=(b(i)-b(j-1))/(x(i)-x(j-1));
    end;
end;
```

### C.0.16 Bsplines

```
function y = bspline(i,k,xi,x)
%-----------------------------------------------------
% Questa funzione valuta la i-esima B-spline
% di ordine k, che usa come nodi xi, nel punto x
%-----------------------------------------------------
% valori iniziali per la relazione di ricorrenza
if (k == 1)
  y = zeros(size(x));
  index = find(xi(i) <= x & x < xi(i+1));
  y(index) = 1;
  return
end
val = (xi(i+k) == xi(i+1));

a =(xi(i+k)-x)/(xi(i+k)-xi(i+1)+val)*(1-val);

val = (xi(i+k-1) == xi(i));

b = (x-xi(i))/(xi(i+k-1)-xi(i)+val)*(1-val);

y = a.*bspline(i+1,k-1,xi,x)+b.*bspline(i,k-1,xi,x);

% continuita' sull'estremo destro dell'intervallo
index2 = find(x == max(xi) & i == length(xi)-k);
y(index2) = 1;
```

*Nota.* Le ultime due righe, servono a rendere continue a destra le B-spline e in particolare quando si vuole costruire una base con nodi multipli con molteplicità pari all'ordine. Si veda per questo il paragrafo 5.7.2 in cui i nodi d'interpolazione sono scelti così da soddisfare alle condizioni di Carrasso e Laurent (cfr. [9]).

### C.0.17 FFT

```
function [ff]=myFFT(f,p)
%-------------------------------------------
% Dato un vettore f di lunghezza n=p*q
% determina il vettore ff che rappresenta
% la FFT di f (algoritmo tratto dal riferimento
% [1], ovvero K. E. Atkinson pag. 181 e ss)
%-------------------------------------------

disp('Trasformata di Fourier tradizionale'); n=length(f);
```

```
for j2=1:n,
    s=0;
    for j1=1:n,
        s=s+f(j1)*exp(-2*pi*i/(n*j2*(j1-1)));
    end
    f1(j2)=s/n;
end

disp('Trasformata di Fourier veloce'); p=2; q=n/p;

for k=1:n,
    s1=0;
    for l=1:p,
        s=0;
        for g=1:q,
          w=exp(2*pi*i*k*(g-1)/q);
          s=s+w*f(l+p*(g-1));
       end
       w1=exp(2*pi*i*k*(l-1)/n);
       s1=s1+(s/q)*w1;
   end
   ff(k)=s1/p;
end
```

### C.0.18 Derivazione numerica

```
function mydiff(x0)
%---------------------------------------
% Calcola la derivata dell'esponenziale
% in x0 con il rapporto incrementale
% e con differenze finite centrali
% per 50 differenti valori del passo.
% Calcola l'errore relativo dei due
% metodi e li plotta.
%---------------------------------------

% valore esatto della derivata
f1xesatta=exp(x0);

for index=1:50
%---------------------------------------
% METODO 1: differenze finite in avanti
%---------------------------------------
% passo
```

```
    h=2^(-index);
% punto "x"
    x=x0+h;
% rapporto incrementale
    f1x(index)=(exp(x)-exp(x0))/h;
% errore relativo 1
    relerr1(index)=abs(f1xesatta-f1x(index))/abs(f1xesatta);
    fprintf('\n \t [x]: %5.5f [h]: %2.2e',x,h);
    fprintf('[rel.err.]: %2.2e',relerr1(index));
%-------------------------------------
% METODO 2: differenze finite centrali
%-------------------------------------
% punti di valutazione
    xplus=x+h; xminus=x-h;
% approssimazione con differenze finite centrali
    f1x(index)=(exp(xplus)-exp(xminus))/(2*h);
% errore relativo 2
    relerr2(index)=abs(f1xesatta-f1x(index))/abs(f1xesatta);
    fprintf('\n \t [x]: %5.5f [h]: %2.2e',x,h);
    fprintf('[rel.err.]: %2.2e',relerr2(index));
end

% plot degli errori
semilogy(1:50,relerr1,'r-+', 1:50,relerr2,'k-o');
return
```

### C.0.19 Quadratura

```
function [integral,nv]=simp_ada(aa,bb,epss,f)
%-----------------------------------------------------------------
%  Calcolo di un integrale con il metodo di Simpson adattativo
%-----------------------------------------------------------------
% Inputs
% aa,bb: estremi dell'intervallo d'integrazione
% epss: tolleranza richiesta
% f: la funzione da integrare
%
% Outputs
% integral: valore approssimato dell'integrale
% nv: numero di valutazioni di funzioni
%-----------------------------------------------------------------
NMAXL=100;        % numero massimo livelli
integral=0;       % valore approssimato dell'integrale
max_err=0;        % errore commesso

ff=input('Fattore dilatazione tolleranza = '); i=1;
```

```
l(i)=1;           % contatore numero livelli
tol(i)=ff*epss;
%----------------------------Plot della funzione integranda
x=aa:.01:bb; for j=1:length(x),
    y(j)=f(x(j));
end; plot(x,y);
title('Metodo di Simpson adattativo');
hold on;
%------------------------------------------------------------
a(i)=aa; b(i)=bb;
h(i)=(b(i)-a(i))/2;
min_h=h(i);
m(i)=a(i)+h(i);
fa(i)=f(a(i));
fb(i)=f(b(i));
fm(i)=f(m(i));
s(i)=h(i)/3*(fa(i)+4*fm(i)+fb(i));
nv=3;     % Valutazioni di funzione
while( i > 0),
% ---------------  PLOT dei punti di integrazione
      p(1)=a(i); p(2)=a(i)+h(i)/2; p(3)=a(i)+h(i);
      p(4)=a(i)+1.5*h(i); p(5)=a(i)+2*h(i);
      plot(p,zeros(5,1),'r+');
      fp(1)=f(a(i));fp(2)=f(a(i)+h(i)/2);fp(3)=f(a(i)+h(i));
      fp(4)=f(a(i)+1.5*h(i));fp(5)=f(a(i)+2*h(i));
      plot(p,fp,'g+');
      nv=nv+2;
% ---------------------------------------------
      fd=f(a(i)+h(i)/2);
      fe=f(a(i)+3/2*h(i));
      ss1=h(i)/6*(fa(i)+4*fd+fm(i));
      ss2=h(i)/6*(fm(i)+4*fe+fb(i));
%---------------------------
%  Salvo i dati del livello
%---------------------------
      t1=a(i); t2=fa(i); t3=fm(i);  t4=fb(i);
      t5=h(i); t6=tol(i); t7=s(i); t8=l(i);
      i=i-1;
      if (abs(ss1+ss2-t7) < t6)
         max_err=abs(ss1+ss2-t7);
         integral=integral+ss1+ss2;
         if( t5 < min_h),
             min_h=t5;
         end;
      elseif (t8 >= NMAXL)
          disp('Superato il livello max. STOP! ');
          break;
      else
```

```
% Meta‘ Intervallo dx .
         i=i+1;
         a(i)=t1+t5;
         fa(i)=t3; fm(i)=fe; fb(i)=t4;
         h(i)=t5/2;
         tol(i)=t6/2;
         s(i)=ss2;
         l(i)=t8+1;
%% Meta‘ Intervallo sx.
         i=i+1;
         a(i)=t1;
         fa(i)=t2; fm(i)=fd; fb(i)=t3;
         h(i)=h(i-1);
         tol(i)=tol(i-1);
         s(i)=ss1;
         l(i)=l(i-1);
      end;
end;
disp('Valore approssimato dell''integrale'); integral
legend('r+','Punti integr.','g+','Valore funzione'); hold off;
disp('Errore max. = ');
max_err disp('Minimo step usato '); min_h
return
```

```
function [I,errest,x] = trapadatt(func,a,b,tol,varargin)
%-------------------------------------------------------------
%  Calcolo di un integrale con
% il metodo dei trapezi adattativo
%-------------------------------------------------------------
% Inputs
% func: la funzione da integrare
% a,b: estremi dell'intervallo d'integrazione
% tol: tolleranza richiesta
%
% Outputs
% I: valore approssimato dell'integrale
% errest: stima d'errore
% x: vettore contenente i nodi usati
%-----------------------------------------------
if (nargin == 4)
  n = 2;
else
  n = varargin{1};
end
h = (b-a)/(n-1);
x = linspace(a,b,n)';
```

```
x(2:end-1) = x(2:end-1)+2*(rand(size(x(2:end-1)))-0.5)*h/10;

weight = h*[0.5,ones(1,length(x)-2),0.5];
I = weight*feval(func,x);
n = 2*n-1; h = h/2; x = linspace(a,b,n)';

x(2:end-1)=x(2:end-1)+2*(rand(size(x(2:end-1)))-0.5)*h/10;

weight=h*[0.5,ones(1,length(x)-2),0.5];

I2 = weight*feval(func,x); errest = abs(I2-I)/2;

if (errest < tol)
  I = I2;
else
  [Il,errestl,xl] = trapadatt(func,a,a+(b-a)/2,tol/2,n);
  [Ir,errestr,xr] = trapadatt(func,a+(b-a)/2,b,tol/2,n);
  I = Il+Ir;
  errest = errestl+errestr;
  x = union(xl,xr);
end

return
```

Questo M-file è un'implementazione dell'esercizio 55.

```
clear; format long

tipoFormula=input('Quale formula: (T) trapezi, (S) Simpson o (G)
gaussiana ? ','s'); n=input('Numero di punti della suddivisione ');
a=0; b=2*pi; h=(b-a)/n;
x=linspace(a,b,n+1); %punti equispaziati.
xx=a:0.01:b; yy=funQ(xx); realValue=quadl(@funQ,a,b,1.e-6); switch
tipoFormula
    case {'T'}
        fTc=funQ(x);
        plot(xx,yy,'-.g',x,fTc,'o');
        fTc(2:end-1)=2*fTc(2:end-1);
        ValTc=0.5*h*sum(fTc)
        title('Quadratura composita con i trapezi e relativi nodi');
        disp('Errore assoluto')
        erroreT=abs(realValue-ValTc)
    case {'S'}
        fSc=funQ(x);
        plot(xx,yy,'-.g',x,fSc,'ob'), hold on
```

```
        fSc(2:end-1)=2*fSc(2:end-1);
        ValSc=h*sum(fSc)/6;
        x=linspace(a+h/2,b-h/2,n);
        fSc=funQ(x);
        plot(x,fSc,'or')
        ValSc=ValSc+2*h/3*sum(fSc)
        title('Quadratura composita di Simpson e relativi nodi');
        disp('Errore assoluto')
        erroreT=abs(realValue-ValSc)
        hold off
    case {'G'}
        y1=x(1:end-1)+h/2*(1-1/sqrt(3));
        plot(xx,yy,'-.g',y1,funQ(y1),'ok'); hold on
        y=[y1 y1+h/sqrt(3)];
        fGc=funQ(y);
        plot(xx,yy,'-.g',y,fGc,'or')
        ValGc=0.5*h*sum(fGc)
        title('Quadratura composita con formula di Gauss e relativi nodi');
        disp('Errore assoluto')
        erroreT=abs(realValue-ValGc)
        hold off
    otherwise
        error('Formula sconosciuta')
end
```

### Tecnica di Romberg

```
function int_value=romberg(f,a,b,n)
%-------------------------------------------------
% Questa M-function implementa il metodo di Romberg
% per la quadratura numerica partendo dalla
% formula dei trapezi composta (mediante la
% formula (6.61)).
%-------------------------------------------------
% Inputs
% f: funzione d'integrazione
% n: numero livelli
% a,b: estremi intervallo d'integrazione
%
% Output
% int_value: valore dell'integrale
%-----------------------------------------
T=zeros(n,n);
%---------------
% Prima colonna
```

```
%---------------
for i=1:n,
  m=2^(i-1); h=(b-a)/m;
  x=linspace(a,b,m+1);
  ff=f(x);
  T(i,1)=(h/2)*(ff(1)+ff(end)+2*sum(ff(2:end-1)));
end;
%--------------------
% Colonne successive
%--------------------
for k=2:n,
  for i=k:n,
    T(i,k)=(4^(k-1)*T(i,k-1)-T(i-1,k-1))/(4^(k-1)-1);
  end;
end;

disp('Tabella di Romberg'); T

int_value=T(n,n);
return
```

# Bibliografia


[1] K. E. Atkinson, *An Introduction to Numerical Analysis*, Second Edition, Wiley, New York, 1989.

[2] R. Bevilacqua, D. Bini, M. Capovani e O. Menchi *Metodi Numerici*, Zanichelli, 1992.

[3] J.-P. Berrut, Lloyd N. Trefethen, *Barycentric Lagrange Interpolation*, SIAM Rev. 46(3) (2004), pp. 501-517.

[4] V. Comincioli, *Analisi numerica: metodi, modelli, applicazioni. E-book*, Apogeo, 2005.

[5] V. Comincioli, *Analisi numerica. Complementi e problemi*, McGraw-Hill Companies, 1991.

[6] M. Conrad e N. Papenberg, *Iterative Adaptive Simpsons and Lobatto Quadrature in Matlab*, TR-2008-012 Mathematics and Computer Science, Emory University, 2008.

[7] P. J. Davis, *Interpolation & Approximation*, Dover Publications Inc., New York, 1975.

[8] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[9] S. De Marchi, *Funzioni splines univariate*, Forum Ed. Udinese, Seconda ed., 2001 (con floppy).

[10] G. Farin, *Curves and Surfaces for CAGD: A Practical Guide*, Third Edition, Academic Press, San Diego, 1993.

[11] Gautschi, W., Inglese, G., Lower bounds for the condition numbers of Vandermonde matrices, *Numer. Math.*, 52(3) (1988), pp. 241-250.

[12] G. Golub, Charles F. Van Loan *Matrix computation*, The Johns Hopkins University Press, Terza Edizione, 1996.

[13] D. Greenspan, V. Casulli *Numerical Analysis for Applied Mathematics, Science and Engineering*, Addison-Wesley, 1988.

[14] Higham, N. J., The Scaling and Squaring Method for the Matrix Exponential Revisited, *SIAM J. Matrix Anal. Appl.*, 26(4) (2005), pp. 1179-1193.

[15] E. Isaacson, H. Bishop Keller, *Analysis of Numerical Methods*, John Wiley & Sons, New York, 1966.

[16] G. G. Lorentz, *Bernstein Polynomials*, Chelsea Publishing Company, New York, 1986.

[17] Moler, C. B. and C. F. Van Loan, Nineteen Dubious Ways to Compute the Exponential of a Matrix, *SIAM Review 20*, 1978, pp. 801-836.

[18] G. Monegato *Elementi di Calcolo Numerico*, Levrotto&Bella, Torino, 1995.

[19] A. Quarteroni, F. Saleri *Introduzione al Calcolo Scientifico*, Esercizi e problemi risolti in Matlab, Terza Ed., Springer-Verlag, Milano, 2006.

[20] A. Quarteroni, R. Sacco e F. Saleri *Matematica Numerica*, Seconda Ed., Springer-Verlag, Milano, 2004.

[21] T. J. Rivlin, *An Introduction to the Approximation of Functions*, Dover Publications Inc., New York, 1969.

[22] J. Stoer, Bulirsch *Introduction to Numerical Analysis* Ed. Springer-Verlag, Berlin, 1980.